Description et simulation d'un système de contrôle de passage à niveau en signal

Bruno Dutertre, Paul Le Guernic

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1437

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

DESCRIPTION ET SIMULATION D'UN SYSTEME DE CONTRÔLE DE PASSAGE A NIVEAU EN SIGNAL

Bruno DUTERTRE
Paul Le GUERNIC

Mai 1991

RR_1437

IRISA INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX
FRANCE
Téléphone : 99.36.20.00
Télex : UNIRISA 950 473F
Télécopie : 99.38.38.32

Description et simulation d'un système de contrôle de passage à niveau en SIGNAL

Bruno Dutertre, Paul Le Guernic

IRISA/INRIA, Campus de Beaulieu
35042 RENNES CEDEX, FRANCE

Publication Interne n° 580 - mars 1991 - 66 pages

Programme 1

Résumé

Dans ce document, nous utilisons le langage SIGNAL pour décrire un système de contrôle et de commande automatique. Une méthode de conception modulaire qui s'appuie sur les principes du langage, approche synchrone du temps et programmation par équations, est présentée. Le système décrit peut ainsi être aisément plongé dans différents contextes, en particulier, il peut être facilement simulé en l'intégrant dans un environnement de simulation lui aussi décrit en SIGNAL.

Description and simulation in SIGNAL of a rail road crossing control system

Abstract

In this report, we use the SIGNAL language to describe an automatic control system. A modular design method, based on the language concept of synchronous programming with equations, is proposed. Hence, the system can easily be embedded in various contexts. Particularly, it can be simulated using a simulation environment also described with SIGNAL.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(U.R.A. 227)
UNIVERSITE DE RENNES I I.N.S.A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(UNITE DE RECHERCHE DE RENNES)

Table des Matières

Chapitre Premier

Introduction

Avec la modernisation de l'appareil industriel, les systèmes de contrôle et de commandes automatiques se répandent dans les domaines de la robotique, des ateliers flexibles, des automates programmables ... Parallèlement, les besoins en logiciels de commande adaptés à ces systèmes, ainsi qu'en outils de conception de ces logiciels, s'accroissent.

Divers formalismes (Réseaux de Petri [15] [1] , STATECHARTS[11], réseaux de files d'attente [14] ...) peuvent modéliser ces systèmes automatiques et permettent, par les possibilités de raisonnement dont ils disposent, d'en étudier certaines propriétés. Cependant, toutes les caractéristiques ne sont pas accessibles par le raisonnement, et ne peuvent donc pas être prouvées. Pour les examiner, la simulation apparaît donc comme un complément nécessaire à tous ces formalismes. Elle permet, d'autre part, de vérifier la correction du modèle et des spécifications.

A partir d'un exemple [1], nous montrons comment le langage temps réel SIGNAL peut constituer un outil de description et de simulation de systèmes automatiques [2]. Ce document illustre une démarche méthodologique pour la conception de tels systèmes qui s'appuie sur les principes du langage : gestion du temps à un niveau logique dans le cadre de l'approche synchrone et conception modulaire grâce à une programmation par équations.

Les spécifications initiales de la simulation sont données en III.1 . La fonction du système est la surveillance des mouvements de trains sur une portion de voie ferrée ; la commande, en temps réel et de manière sûre, d'un passage à niveau ; ainsi que la production d'informations destinées à un site de contrôle. Cet exemple est suffisamment réaliste pour constituer un support valable à l'étude des possibilités de conception et de simulation qu'offre SIGNAL. Il présente certaines

[1] Cet exemple est emprunté au projet ESPRIT Descartes.

[2] Ce travail est réalisé en collaboration avec CGE Ldm, dans le cadre du projet MRT SOSIE [9].

caractéristiques importantes de ces systèmes :

- Les contraintes temporelles ne se limitent pas à un problème de vitesse d'éxécution.
- Les problèmes de fiabilité de certains composants sont pris en compte.

La conception de la simulation s'est déroulée en trois étapes :

- Un programme SIGNAL, répondant aux spécifications et qui pourrait donc être directement implémenté sur le site, a été réalisé. Dès cette étape, des propriétés fonctionnelles et temporelles (absence d'interbloquage, synchronisation ...) sont prouvées par le compilateur du langage.
- Le programme précédent est plongé dans un environnement de simulation, lui aussi décrit en SIGNAL. Les propriétés qui échappent aux capacités de raisonnement formel du langage peuvent alors être vérifiées.
- L'ensemble est ensuite muni d'une interface graphique qui permet la visualisation des phénomènes, ainsi que l' interaction de l'expérimentateur.

Nous présentons dans la suite, les fondements du langage SIGNAL puis les trois étapes de la réalisation.

Chapitre II

Le langage Signal

SIGNAL [8] est un langage de programmation qui permet la description de systèmes temps-réel. Il fait partie de la famille des langages synchrones qui se distinguent des langages classiques par la manière dont ils considèrent le temps.

SIGNAL est un langage descriptif à flots de données : un processus SIGNAL est un ensemble d'équations portant sur des séquences de valeurs (les signaux). La composition de tels processus produit, de manière naturelle, de nouveaux ensembles d'équations. SIGNAL est conçu incrémentalement autour d'un langage noyau que nous introduisons en II.3.1 et II.3.2. Il est complété par un ensemble d'opérateurs définis formellement sur ce noyau. Certaines de ces extensions, qui seront utilisées par la suite, sont présentées en II.3.4.

La programmation en SIGNAL consiste donc à composer des processus de base (les opérateurs élémentaires du langage), créant ainsi de nouvelles entités qui pourront elles-mêmes être composées.

II.1 Le temps logique

Les systèmes temps-réels [15] manipulent des données mais doivent aussi prendre en compte des contraintes temporelles. La validité des résultats dépend non seulement de leur correction logique mais aussi des instants auxquels ces résultats sont produits. De tels systèmes sont appelés parfois *systèmes réactifs* [9], c'est à dire qu'ils interagissent de façon répétitive avec leur environnement.

Les langages ou formalismes destinés à la description de ces systèmes doivent donc disposer de moyens d'appréhender le temps. On distingue, dans ce domaine, deux approches distinctes : les langages asynchrones et les langages synchrones.

Un système asynchrone a une vision chronométrique du temps : les durées

des phénomènes, en particulier des calculs et des communications, doivent être prises en compte. Les événements sont liés logiquement par des relations de simultanéité et de précédence, mais aussi quantitativement par les délais qui existent entre eux. Les difficultés posées par cette approche résident dans son indéterminisme, l'enchaînement des tâches pouvant être imprévisible.

Pour remédier aux insuffisances du modèle asynchrone, d'autres langages ont été développés : le langage impératif ESTEREL [3] [4], les langages descriptifs LUSTRE [6] et SIGNAL [11] sont spécialement adaptés à la description de systèmes réactifs. Visant à rendre la programmation indépendante d'une architecture de mise en œuvre particulière, ils font l'hypothèse que les durées des actions internes (calculs, communications ...) sont nulles. Cette hypothèse permet de se placer dans un cadre formel, où l'étude des propriétés du programme (déterminisme, étreinte fatale ...) n'est pas liée à des vitesses d'éxécution, à des débits de communications.

Dans la conception des systèmes synchrones, la notion de durée n'intervient donc plus ; elle est prise en compte au niveau de la mise en œuvre qui doit être une approximation correcte de l'hypothèse de temps de calcul nul.

Dans le cadre synchrone, deux événements ne sont plus liés que par les relations de précédence et de simultanéité, le temps n'est que l'expression de relations *logiques* entre événements. On ne se réfère pas à un temps global unique mais on s'intéresse aux contraintes qui lient les événements les uns aux autres.

II.2 Les signaux

Le langage SIGNAL [1] est un langage à flots de données synchronisés ; il manipule des séquences typées et datées de valeurs : les *signaux*.

Un signal est une suite infinie de valeurs, les instants de présence de ces valeurs, relativement aux autres signaux observés, étant définis par l'*horloge* du signal.

[1] On trouvera une présentation complète des fondements théoriques du langage dans [11] et [2].

Exemple :

X :	2			5	3			1	....
Y :			1	1			4	0	....

Le signal **X** est présent aux instants 1, 3, 4, 6, ..., le signal **Y** aux instants, 2, 3, 5, 6, ... Ces instants sont relatifs à l'*environnement* considéré (i.e. à l'ensemble des signaux observés) ; il n'y a pas de référentiel de temps global.

Deux signaux qui sont présents aux mêmes instants pour tout environnement sont dits *synchrones* ; une classe d'équivalence de signaux synchrones est appelée l'horloge de ces signaux.

En SIGNAL, les relations temporelles entre les signaux sont décrites par des contraintes entre leurs horloges. On définit ainsi une relation d'ordre sur les horloges :

$$H_1 \leq H_2$$

exprime qu'à chaque instant de présence d'un signal S_1 d'horloge H_1, tout signal S_2 d'horloge H_2 est aussi présent. Dans ce cas, on dira que S_2 est plus fréquent que S_1.

II.3 Les processus

Les processus de SIGNAL sont des systèmes d'équations sur les signaux, ils expriment à la fois des relations entre les valeurs de ces signaux et des relations temporelles entre leurs horloges.

Pour définir les processus, SIGNAL propose :

- un ensemble d'opérateurs de base, à partir desquels sont contruits les processus élémentaires du langage,
- une opération de composition des processus,
- un mécanisme d'abstraction, les *modèles de processus* , qui permet une programmation modulaire.

II.3.1 Les processus élémentaires de SIGNAL

On distingue, parmi les opérateurs de base de SIGNAL, ceux qui expriment des équations entre signaux synchrones : *expressions fonctionnelles* et *retard* de ceux qui décrivent des relations entre signaux d'horloges différentes : *condition* et *fusion.*

Les expressions fonctionnelles

Les fonctions définies sur les types du langage (par exemples les fonctions arithmétiques ou booléennes), sont étendues canoniquement aux signaux. Le signal (Y_t) défini par :

$$Y_t = f(X1_t, X2_t, \ldots, Xn_t) \quad t \in T$$

où f est une fonction instantanée, s'écrit en SIGNAL :

$$Y := f(X1, X2, \ldots, Xn)$$

Dans une telle expression, les valeurs des signaux Y, $X1$, ..., Xn sont disponibles aux mêmes instants, la fonction ne consomme pas de temps; les signaux Y, $X1$, ..., Xn sont donc synchrones.

Le retard

Pour accéder aux valeurs passées d'un signal, on peut définir :

$$ZX_t = X_{t-1} \quad t \in T$$

ce qui s'écrit en SIGNAL :

$$ZX := X \ \$1$$

Les signaux ZX et X sont là aussi synchrones, ils sont disponibles aux mêmes instants, et à chacun de ces instants, ZX possède la valeur précédente de X. La valeur initiale $v0$ de ZX est spécifiée par :

$$ZX \ \textbf{init} \ v0$$

Les expressions conditionnelles

Un opérateur de SIGNAL permet de sous-échantillonner un signal par une condition booléenne :

$$Y := X \textbf{ when } B$$

Les occurrences du signal **Y** sont les occurrences **X** simultanées à une occurrence *vraie* du signal booléen **B**.

Exemple :

X :	x_1	x_2		x_3	x_4	x_5	....
B :	F	V	F		V	F	....
Y :		x_2			x_4		....

Y est donc à la fois moins fréquent que **X** et que **B**.

La fusion

Le dernier opérateur de base de SIGNAL fusionne, de manière déterministe, deux signaux de même type :

$$Z := X \textbf{ default } Y$$

La valeur de **Z** est égale à la valeur de **X** quand **X** est présent, à la valeur de **Y** quand **X** est absent et **Y** présent. **Z** est par conséquent plus fréquent que **X** et que **Y**.

II.3.2 La composition de processus

Si $P1$ et $P2$ sont deux processus, la *composition* de $P1$ et $P2$ définit un nouveau processus noté :

$$P1 \mid P2$$

où les noms communs à $P1$ et $P2$ référencent les mêmes signaux. $P1$ et $P2$ communiquent par l'intermédiaire de ces signaux communs.

L'opérateur de composition ainsi défini est associatif et commutatif, il réalise l'union des deux systèmes d'équations décrits par $P1$ et $P2$.

On peut réduire les possibilités de communications d'un processus en masquant des signaux :

$$P \ / \ a_1, \ldots, a_n$$

définit un processus identique à P mais dans lequel les signaux $a_1, \ldots, a_n$ sont masqués. Ils restent locaux et ne peuvent pas servir de voie de communication.

II.3.3 Les modèles de processus

SIGNAL offre un mécanisme d'abstraction : les *modèles de processus* qui permet une programmation modulaire en associant un nom et une interface à un système d'équations (le corps du modèle).

La syntaxe de déclaration d'un modèle est la suivante :

process *NOM* =
  (*paramètres*)
  { ? *signaux d'entrées*
  ! *signaux de sorties* }
    (| *corps du processus*
    |)
where
  déclarations locales
end

L'interface du processus définit les paramètres d'initialisation, les signaux d'entrée et de sortie du modèle. Tous les signaux qui ne sont pas nommés dans l'interface sont invisibles de l'extérieur du processus ; ils doivent être déclarés localement et masqués.

Le corps du processus est une expression définie à partir des opérateurs de base et d'éventuels modèles de processus locaux.

Un exemplaire ou *instance* d'un modèle de processus, noté :

NOM(*définition des paramètres*),

définit un processus identique au corps du modèle, qui ne peut communiquer que par les signaux cités dans l'interface.

II.3.4 Les opérateurs dérivés

Autour du noyau de SIGNAL, est défini un ensemble d'opérateurs dérivés. Nous ne citerons ici que ceux qui seront utilisés dans la suite ; pour une description plus complète, se reporter à [12] ou à [7].

Les opérateurs dérivés peuvent être classés en deux catégories : les expressions sur signaux, et les expressions sur processus. Les premières enrichissent l'ensemble des opérateurs de base, les secondes définissent de nouvelles façons de combiner les processus.

Opérateurs sur signaux

Sur l'ensemble des opérateurs disponibles en SIGNAL, nous n'utiliserons que les trois suivants :

- **event** X est un signal booléen toujours vrai, de même horloge que $\mathbf{X}$, qui peut donc être assimilé à l'horloge de $\mathbf{X}$.

- l'expression X **cell** B est la mémorisation de $\mathbf{X}$. Quand $\mathbf{X}$ est présent, sa valeur est produite en sortie de la mémorisation , quand la valeur *vrai* est présente en $\mathbf{B}$ et que $\mathbf{X}$ est absent, la précédente valeur de $\mathbf{X}$ est transmise en sortie.

- une expression de la forme **synchro** $X_1, X_2, \ldots, X_n$ est une *synchronisation explicite*, elle spécifie que les signaux $\mathbf{X}_1, \ldots, \mathbf{X}_n$ ont la même horloge.

Voici, à titre d'exemple, comment l'opérateur **cell** peut être défini à partir des opérateurs élémentaires :

L'expression $Y := X$ **cell** B est équivalente au programme suivant :

```
( | Y  := X default ZY
  | ZY := Y $1
  | HY := (event X) default (true when B)
  | synchro {Y, HY}
  | )
```

Les deux premières équations expriment qu'à chaque instant de présence de **Y**, soit **X** est présent et la valeur de **Y** est égale à celle de **X** ; soit **X** est absent et la valeur de **Y** est celle de **ZY**, c'est à dire la valeur précédente de **Y**. Il est clair que cette dernière est la plus récente valeur de **X**.

Il reste à définir à quels instants **Y** est présent. C'est ce que font les deux dernières équations. La troisième définit le signal horloge **HY** comme la fusion de l'horloge de **X** et de l'horloge des valeurs *vraies* de **B**. La quatrième équation indique que **Y** est synchrone à **HY**, autrement dit **HY** est l'horloge de **Y**. On obtient donc le résultat souhaité : **Y** est présent quand **X** est présent ou quand **B** est *vrai*.

Opérateurs sur processus

Il existe, en SIGNAL, des opérateurs, extensions du masquage, qui permettent de décrire les communications entre processus.

Dans la suite, nous utiliserons la forme duale du masquage :

$$P \;!!\; a_1, \ldots, a_n$$

est le processus P dans lequel toutes les sorties autres que $a_1, \ldots, a_n$ sont masquées.

Nous utiliserons aussi les renommages, qui permettent d'établir des connexions entre processus :

$$P \;?\; a : x \;!\; b : y$$

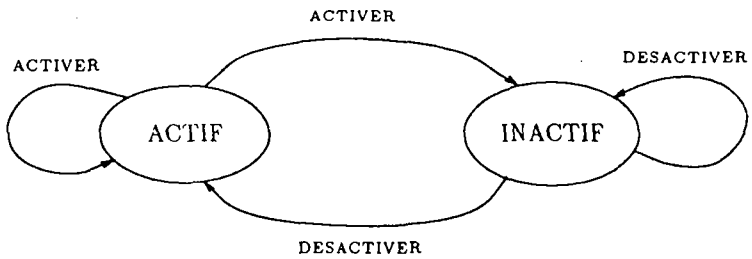
est le processus égal à P dans lequel l'entrée a est renommée en x, la sortie b en y. Ce nouveau processus peut ainsi communiquer avec un autre qui possède des signaux de nom x et y.

II.3.5 Exemple

Nous illustrons les diverses notions abordées par un exemple extrait de l'application.

Considérons un organe, pouvant être ACTIF ou INACTIF, contrôlé par deux commandes ACTIVER et DESACTIVER.

Son comportement peut être décrit par l'automate suivant :



Nous désirons maintenant construire un processus SIGNAL équivalent.

La suite des états successifs de l'automate peut être représentée par un signal booléen ; ici, on a choisi la valeur *vrai* pour ACTIF, la valeur *faux* pour INACTIF.

Si on ne fait pas l'hypothèse que les deux commandes sont mutuellement exclusives[2], on peut associer à chacune d'elle un signal horloge pur[3] :

- Le signal **ACTIVER** représente la suite des commandes d'activation,
- le signal **DESACTIVER**, la suite des désactivations.

Appelons **SACTIF** le signal booléen des états successifs. La présence d'une commande **ACTIVER** (resp. **DESACTIVER**) produit un échantillon *vrai* (resp. *faux*) sur **SACTIF**. On choisit, quand deux commandes contradictoires sont reçues simultanément, de privilégier **ACTIVER** ; **SACTIF** est alors défini par la relation :

SACTIF := ACTIVER **default** (**not** DESACTIVER)

Avec cette définition, l'état n'est disponible que lors de la réception d'une commande ; on souhaiterait pouvoir le consulter plus souvent. Pour cela, il faut sur-échantillonner le signal **SACTIF**. Voici une manière de procéder :

[2]Dans le cas contraire, un seul signal de commande possédant les valeurs activer et désactiver aurait suffit.

[3]Le type **event** est celui des signaux horloges, ces signaux peuvent être assimilés à des signaux booléens qui possèdent toujours la valeur *vrai*.

```
  | ACTIF  := SACTIF default ZACTIF
  | ZACTIF  := ACTIF $1
```

Une valeur du signal **ACTIF** est égale à la plus récente valeur du signal **SACTIF**.

L'horloge du signal **ACTIF** est indéfinie, nous savons seulement qu'elle est supérieure à celle de **SACTIF**. L'extérieur doit spécifier cette horloge en indiquant à quels instants l'état doit être consulté. A cet effet, on a ajouté une entrée au processus : l'horloge de consultation **H**.

La définition complète du modèle de processus **etat** est alors la suivante :

```
process ETAT =
     { ? event ACTIVER, DESACTIVER, H
       ! logical ACTIF, ZACTIF}

     (| SACTIF := ACTIVER default (not DESACTIVER)
      | ACTIF  := SACTIF default ZACTIF
      | ZACTIF := ACTIF $1
      | HACTIF := (event SACTIF) default H
      | synchro {ACTIF, HACTIF}
      |)
where
     event HACTIF
end
```

Le chronogramme suivant schématise une évolution possible d' **etat**, avec le signal **ZACTIF** initialement *faux*.

ACTIVER :	—				—	—		—	....
DESACTIVER :			—				—	—	....
SACTIF :	*V*		*F*		*V*	*V*	*F*	*V*	....
H :		—		—	—				....
ACTIF :	*V*	*V*	*F*	*F*	*V*	*V*	*F*	*V*	....
ZACTIF :	*F*	*V*	*V*	*F*	*F*	*V*	*V*	*F*	....

Un exemple d'utilisation d'**etat** est donné dans le modèle **interrupteur** ci-dessous :

```
process INTERRUPTEUR =
      (logical INITIAL)
      { ? event OUVRIR, FERMER, ENTREE
        ! event SORTIE}

      ( | {ACTIF, ZACTIF}    :=
              ETAT() {OUVRIR, FERMER, ENTREE}
        | SORTIE := ENTREE when ZACTIF
        | )
where
      logical ACTIF, ZACTIF init INITIAL
end
```

Etat contrôle ici, par l'intermédiaire des commandes **OUVRIR** et **FERMER**, la transmission du signal **ENTREE** vers la **SORTIE**. Quand **ZACTIF** est *vrai*, l'**interrupteur** est ouvert, **ENTREE** est transmis ; quand **ZACTIF** est *faux*, l'**interrupteur** est fermé. Le paramètre fixe l'état initial de l'interrupteur.

II.4 La représentation graphique des processus

Le langage SIGNAL se prête , par sa modularié, à une représentation graphique directe en blocs-diagrammes. Un éditeur [5] est disponible dans l'environnement SIGNAL qui permet la construction sous une forme mixte (mélangeant graphiques et textes) des programmes.

Nous en détaillons la structure à l'aide de la figure II.1, qui représente le modèle de processus **interrupteur** décrit précédemment.

Un modèle de processus est représenté par quatres zones rectangulaires distintes qui contiennent respectivement :

- Le nom du modèle (**INTERUPTEUR**),
- la déclaration des paramètres d'initialisation (**logical INITIAL**)
- le corps du processus
- la définition des modèles de processus locaux.

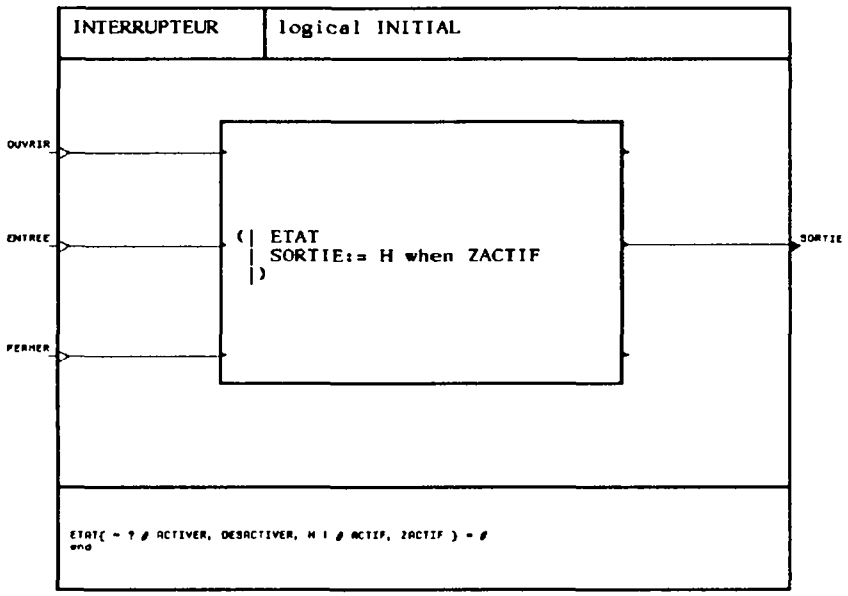


Figure II.1 : la représentation graphique du modèle **interrupteur**

Les signaux d'entrée et de sortie du modèle sont symbolisés par les ports de connexions triangulaires situés sur les bords de la boîte corps de processus.

Chaque boîte interne au corps représente une expression de processus qui peut être soit sous forme textuelle, soit construite récursivement à partir d'autres boîtes. Les signaux non masqués de chacune de ces sous expressions sont représentés par des ports de connexion.

Les liens entre les différents ports indiquent les voies de communications entre les divers sous processus. Ces liens sont donc l'équivalent graphique des renommages : deux ports reliés par un même lien représentent le même signal.

Chapitre III

Le système de contrôle

III.1 Spécification informelle

L'exemple présenté a été tiré du projet ESPRIT Descartes où il illustre les différentes méthodes et outils qui y sont développés pour la conception de systèmes temps-réel.

Il décrit le système de contrôle d'un passage à niveau automatique qui surveille les mouvements des trains sur la portion de voie considérée, commande les barrières et les signaux lumineux de manière sûre, et fournit des informations à un opérateur humain par l'intermédiaire d'un tableau de contôle.

III.1.1 Description de l'environnement

On considère une portion de chemin de fer formée de deux voies sur lesquelles des trains roulent en sens opposés, chacune des voies ayant un sens de circulation prédéfini.

Une route traverse les voies au milieu du segment considéré. Ce croisement est muni d' un passage à niveau automatique comprenant deux barrières et deux signaux lumineux. Les barrières peuvent tomber en panne ou être bloquées par un éventuel véhicule. Les lampes ont trois positions possibles : éteintes, clignotantes ou allumées en continu, et sont supposées totalement fiables.

Des capteurs situés sur les voies fournissent des informations sur les mouvements des trains, d'autres, associés aux barrières signalent la fin des opérations d'ouverture ou de fermeture.

Le système doit interpréter les informations issues de ces différents capteurs pour commander les barrières et les lampes et indiquer, à un tableau de contrôle,

l'occupation des trois secteurs (ouest, central et est) de chacune des voies ainsi que la position et les éventuelles pannes des barrières.

L'ensemble est schématisé par la figure ci-dessous :

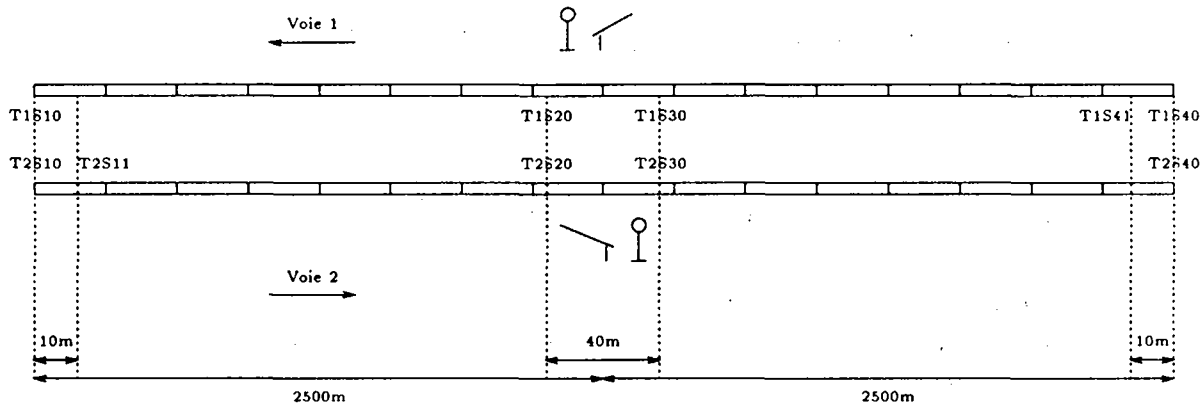


Figure III.1 : Le passage à niveau

Les différents capteurs et commandes par lesquels le système interagit avec son environnement sont présentés dans le tableau suivant :

Nom	Description
T_iS_j	Capteur : émet un signal au passage d'un wagon à la position j de la voie i
B_iSU	Capteur : Signale que la barrière i atteint sa position haute
B_iSD	Capteur : Signale que la barrière i atteint sa position basse
B_iCD	Commande de fermeture de la barrières i
B_iCU	Commande d'ouvertue de la barrière i
L_iCS	Commande de clignotement du feu i
L_iCH	Commande d'allumage continu du feu i
L_iCS	Commande d'extinction du feu i

Le tableau de contrôle fournit les informations suivantes :

Nom	Description
SB_i	Indique la position de la barrière i
EB_iU	Indique une panne de la barrière i détectée lors d'une ouverture
EB_iD	Indique une panne détectée lors d'une fermeture
EB_iC	Fonctionnement correct de la barrière
TS_is	Indique l'occupation de la section s de la voie i

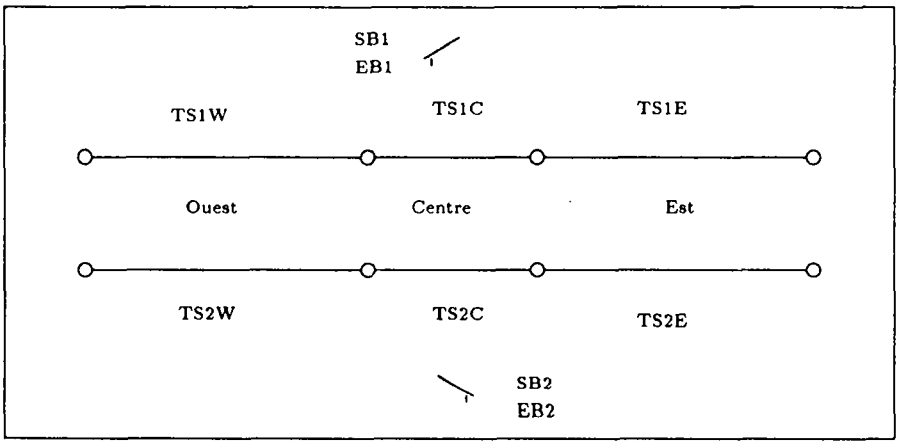


Figure III.2 : Le tableau de contrôle

III.1.2 Les contraintes

La vitesse des trains sur les deux voies est supposée constante et comprise entre 10 et 160 km/h. Un train peut contenir jusqu'à 100 wagons d'une longueur de 10 à 35 mètres. La distance entre deux trains est d'au moins 4000 mètres.

Le passage d'un wagon sur un capteur provoque l'émission d'un signal par ce dernier.

L'objectif du système de contrôle est de commander de manière sûre l'ouverture et la fermeture des barrières, sans que cela entraine des délais d'attente trop long pour les véhicules sur la route. Les contraintes à respecter sont les suivantes :

- Les barrières doivent être fermées, au plus tôt 40 secondes, au plus tard 30 secondes, avant qu'un train n'atteigne le passage à niveau.

- On ne doit pas ouvrir les barrières pour une durée inférieure à 15 secondes.

Les deux barrières disposant d'une mécanique séparée, doivent êtres commandées de manière indépendante. Les ordres d'ouverture ou de fermeture sont suivis d'une réponse du capteur correspondant dans un délai de 5 secondes, sinon, la barrière est en panne.

Les opérations d'ouverture et de fermeture doivent respecter les contraintes suivante :

(1) Pour la fermeture :

- Le système de contrôle doit mesurer la vitesse des trains grâce aux capteurs **T1S40** et **T1S41** de la voie 1, ou **T2S10** et **T2S11** de la voie 2.
- 40 secondes avant que les trains arrivent dans le secteur central, il doit commander la fermeture des barrières et le clignotement des feux.
- Si 5 secondes après cette commande, aucune confirmation de fermeture n'est reçue, une deuxième commande doit être envoyée et une panne signalée au tableau de contrôle.
- Si cette nouvelle commande échoue, la lampe doit être allumée de manière continue.

(2) Pour l'ouverture :

- Quand le dernier wagon d'un train a dépassé le secteur central, et que l'approche d'aucun autre train n'impose que les barrières restent fermées, le système commande l'ouverture des barrières.
- Si cette ouverture échoue, la demande doit être renouvelée après 5 secondes, et une panne signalée.
- Les lampes peuvent être éteintes seulement quand l'ouverture a réussi.

III.2 Description du système de contrôle en SIGNAL

Les concepts du langage SIGNAL permettant une programmation modulaire, la méthode de description adoptée ici procède par décompositions successives en processus qui assurent chacun une fonction du système global.

Le programme obtenu peut ainsi facilement évoluer ; chaque fonction, localisée dans un module particulier, peut être modifiée sans compromettre le fonctionnement des autres composants.

Le système de contrôle du passage à niveau assure deux fonctions représentatives des systèmes temps réel : une fonction de contrôle du procédé physique (ici l'ouverture et la fermeture de barrières) et une fonction d'interfaçage avec un opérateur humain, au travers d'un tableau de contrôle.

Plus précisément, les tâches à effectuer sont les suivantes :

- Fonctions de commande :
  - ouverture et fermeture des barrières,
  - gestion des feux de signalisation.
- Fonctions de surveillance :
  - indiquer l'occupation des différents secteurs de chaque voie,
  - donner la position et signaler les défaillances éventuelles des barrières.

Si la surveillance de l'occupation des voies peut être effectuée indépendamment des commandes, il n'en est pas de même de la détection des pannes. En effet, c'est l'échec d'une commande d'ouverture ou de fermeture qui indique la défaillance des barrières. Il est clair aussi que la commande des feux ne peut pas être dissociée de la commande des barrières.

Par conséquent, on a adopté une décomposition du système de contrôle en deux sous-systèmes :

- Le premier est chargé de surveiller, et de transmettre au tableau de contrôle, l'occupation des voies.
- Le second commande les barrières et les feux, et signale les pannes.

Le processus de contrôle **contrôle_passage** est donc directement décrit comme la composition de deux sous-processus appelés **occupations** et **passage** qui réalisent respectivement chacune de ces deux fonctions.

III.3 L'occupation des voies

L'occupation de chacun des trois secteurs de chacune des deux voies est déterminée à partir des informations issues des capteurs qui délimitent ces secteurs.

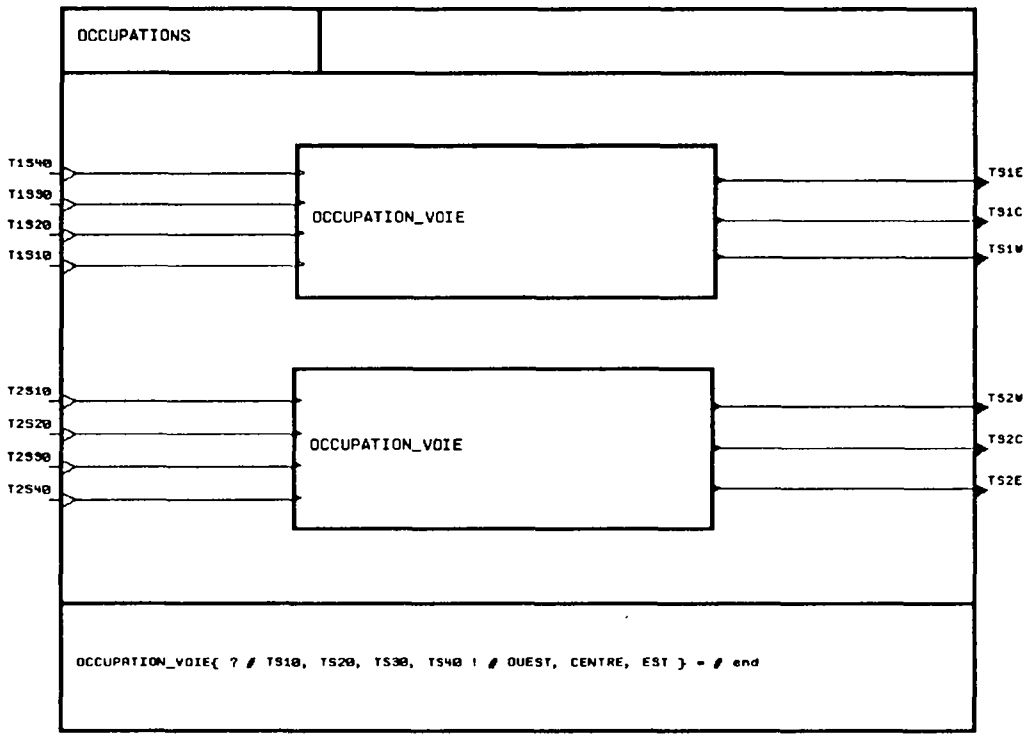


Figure III.3 : le processus **occupations**

Le processus **occupations** qui produit ces informations reçoit donc les signaux $\mathbf{T_iS10}$, $\mathbf{T_iS20}$, $\mathbf{T_iS30}$ et $\mathbf{T_iS40}$ émis par les capteurs au passage des wagons, et transmet les signaux d'occupation des voies : $\mathbf{TS_iW}$ pour le secteur Ouest, $\mathbf{TS_iC}$ pour le secteur Central et $\mathbf{TS_iE}$ pour le secteur Est de la voie i, au tableau de contrôle.

Les deux voies possédant le même nombre et la même disposition de capteurs, peuvent être surveillées par deux processus identiques. **Occupations** est donc décomposé en deux sous-processus similaires, chargés chacun d'une des voies, instance du modèle **occupation_voie** (figure III.3).

De la même façon, **occupation_voie** est formé de trois processus identiques, associés chacun à l'un des secteurs. Ces processus, instances du modèle **occupation_secteur** dont le texte est donné ci-dessous, déterminent l'occupation d'un secteur en comptant le nombre de wagons qui s'y trouvent.

```
process OCCUPATION_SECTEUR =
    { ? event TDEB, TFIN
      ! logical OCCUPE}

    ( | DN := ((0 when TDEB) when TFIN) default
              (1 when TDEB) default (-1 when TFIN)
      | N := ZN + DN
      | ZN := N $1
      | OCCUPE := (true when (ZN = 0)) default
              (false when (N = 0))
      | )
where
    integer N, ZN init 0, DN
end
```

Les signaux d'entrée **TDEB** et **TFIN** sont issus respectivement du capteur de début et de fin du secteur. Le signal **N** est le nombre de wagons situés entre ces deux capteurs. Quand deux wagons entrent et sortent simultanément, **N** est inchangé (**DN** est nul) ; sinon, **TDEB** et **TFIN** augmentent ou diminuent respectivement le signal **N**.

Les valeurs *vrai* du signal de sortie **OCCUPE**, produites quand **N** devient positif (i.e. sa valeur précédente **ZN** devient nulle), indiquent que le secteur devient occupé. Les valeurs *faux* signalent la libération du secteur (**N** devient nul).

Il est important de remarquer que ce procédé ne peut pas être appliqué pour déterminer l'occupation du secteur situé entre les deux premiers capteurs. En effet, pour qu'**occupation_secteur** fonctionne correctement, il faut que la distance entre les deux capteurs **TDEB** et **TFIN** soit supérieure à la longueur de chaque wagon.

III.4 La commande des feux et des barrières

A partir des différents capteurs, le système de contrôle doit commander les feux et les barrières ; c'est à dire, déterminer les instants de début et de fin d'interdiction de circuler sur la route puis, conformément à la procédure donnée par les spécifications (cf. III.1.2), transmettre les commandes correspondantes aux divers agents (feux, barrières, tableau de contrôle).

Les instants de fermeture sont calculés en estimant la vitesse des trains à partir des deux premiers capteurs de chaque voie.

Les instants d'ouverture sont des instants de libération du secteur central, cette information est déjà disponible en sortie du module de surveillance de l'occupation des voies.

Les signaux d'acquittement des opérations d'ouverture et de fermeture sont eux aussi nécessaires (cf III.1.2).

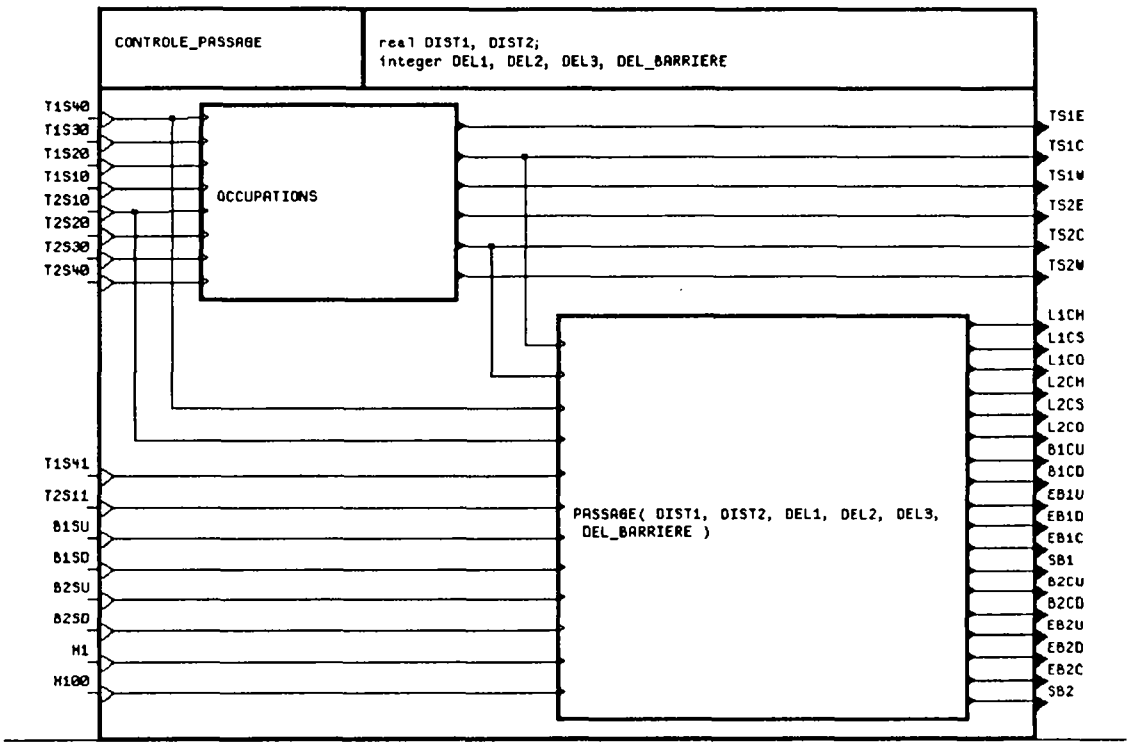


Figure III.4 : le processus **controle_passage**

Le processus de commande (**passage**) reçoit donc les signaux d'entrée (figure III.4) :

- **T1S40** et **T1S41**, issus des deux premiers capteurs de la voie 1.
- **T2S10** et **T2S11**, provenant de la voie 2,
- **TS1C** et **TS2C**, occupations des secteurs centraux,
- **B1SU**, **B1SD**, **B2SU** et **B2SD** signaux d'acquitements émis par les capteurs des barrières.

Pour estimer la vitesse des trains, le système doit chronométrer la durée du parcours entre les deux premiers capteurs **T1**–**T2** de chaque voie. Cette durée pouvant atteindre $1/4$ s à 160 km/h, une estimation raisonnable nécessite des calculs au $1/100$ s. Les autres délais mis en œuvre sont eux de l'ordre de la seconde, c'est pourquoi, **passage** est pourvu de deux horloges : **H100** cadencée au $1/100$ s et **H1** cadencée à la seconde.

Les divers paramètres du processus son extraits des spécifications :

- **DIST1**, distance entre les deux premiers capteurs **T1** et **T2** d'une voie,
- **DIST2**, distance entre **T2** et le passage à niveau,
- **DEL1**, délai maximal entre la fermeture des barrières et l'arrivée d'un train au passage à niveau.
- **DEL2**, durée minimale d'ouverture des barrières.
- **DEL3**, délai maximal entre le passage à un même point de deux wagons successifs,
- **DEL_BARRIERE**, temps maximum de réponse des barrières à une commande.

DEL3 qui sert à détecter la fin des trains (cf III.5.1) est calculé à partir de la vitesse minimale des trains et de la longueur maximale des wagons.

En faisant l'hypothèse que les trains roulent à vitesse constante, ces paramètres suffisent pour, connaissant la durée δ du parcours **T1**–**T2**, calculer le délai Δ avant fermeture des barrières :

$$\Delta = \delta * \frac{\mathbf{DIST1}}{\mathbf{DIST2}} - \mathbf{DEL1}$$

Les sorties du processus **passage** sont les signaux de commandes transmis aux barrières ($\mathbf{B_iCU}$, $\mathbf{B_iCD}$) et aux feux ($\mathbf{L_iCH}$, $\mathbf{L_iCS}$, $\mathbf{L_iCO}$), et les signaux d'informations destinés au tableau de contrôle ($\mathbf{EB_iU}$, $\mathbf{EB_iD}$, $\mathbf{EB_iC}$, $\mathbf{SB_i}$).

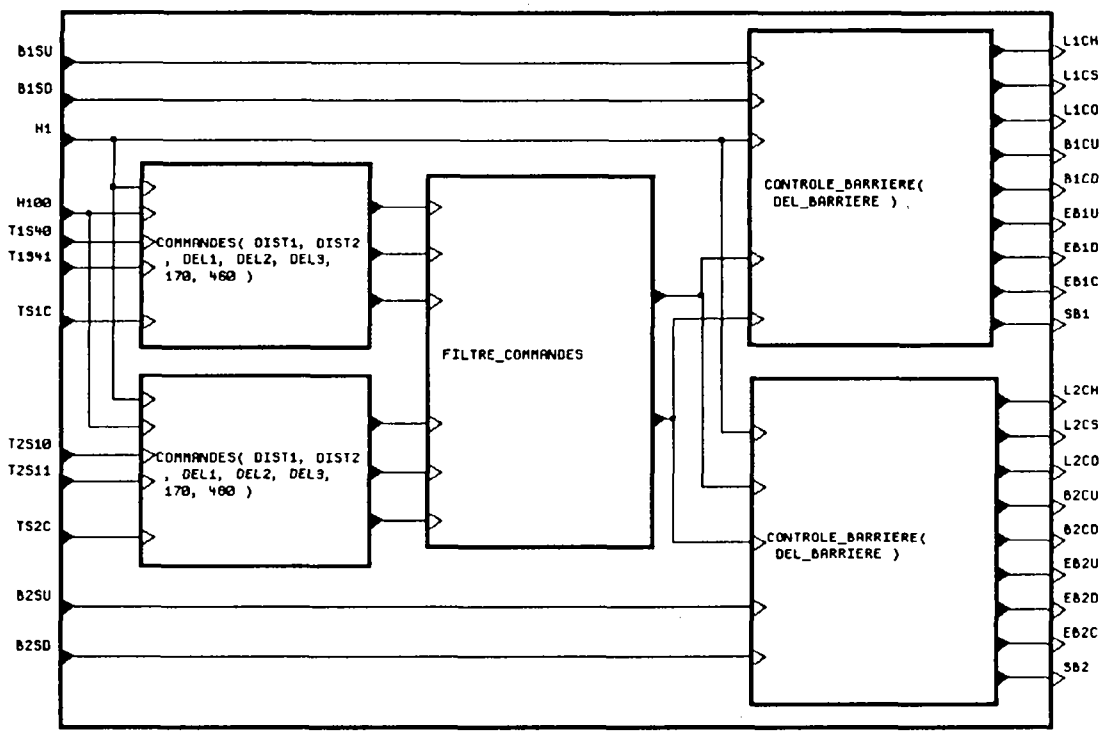


Figure III.5 : le processus **passage**

III.5 Le processus de commande : passage

Comme on l'a vu précédemment, le processus de commande doit, à partir de la circulation des trains sur chaque voie, déterminer quand ouvrir ou fermer les barrières et appliquer la procédure d'ouverture ou de fermeture correspondante.

Les deux voies sont identiques mais indépendantes, les deux ensembles barrière–feu aussi. Cependant la décision d'ouvrir ou de fermer est commune aux deux barrières et doit tenir compte des deux voies.

Une décomposition naturelle du système de commande est donc :

- un module de surveillance des mouvements de trains pour chaque voie,
- un module centralisateur qui calcule les instants d'ouverture et de fermeture,
- un module, chargé d'éxécuter les procédures d'ouverture / fermeture, associé à chaque groupe barrière–feu.

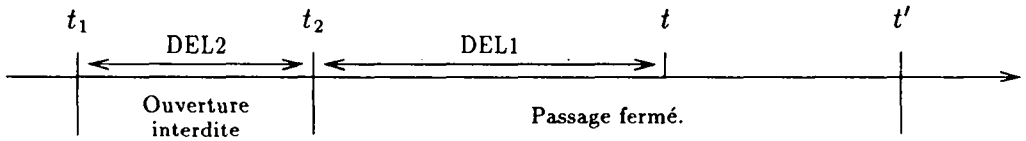
On a donc défini trois modèles de processus : **commandes**, **filtre_commandes** et **controle_barrière** qui décrivent respectivement chacune de ces trois activités et dont les instances composent le processus de commmande (figure III.5).

Cette décomposition a l'avantage de permettre l'extension relativement simple du système. Par exemple, il suffit d'ajouter une instance du modèle **commandes** et de modifier le modèle **filtre_commandes** pour ajouter une voie.

III.5.1 Le processus commandes

Il faut extraire, des informations brutes issues des capteurs, celles qui sont réellement utiles pour commander les barrières.

Considérons un train qui atteint le passage à niveau à l'instant t et le quitte à l'instant t'. Le chronogramme suivant résume les contraintes



De t_2 à t', la circulation sur la route est interdite, les barrières doivent être fermées si elles ne sont pas en panne. Entre t_1 et t_2, les barrières, si elles sont fermées, ne doivent pas être ouvertes, pour respecter la durée d'ouverture minimale (**DEL2**).

Pour décider de l'ouverture ou de la fermeture des barrières, il suffit donc de connaître, pour chaque train circulant sur les voies, les instants t_1, t_2 et t'.

A cet effet, les deux processus **commandes** (figure III.6) surveillent chacun une voie et synthétisent à partir des signaux **TS1** et **TS2** en provenance des deux premiers capteurs et de **TSC**, occupation du secteur central, les trois signaux suivants :

- **T** qui interdit l'ouverture des barrières si elles sont en position basse (horloge des instants t_1),
- **DESCENDRE** qui impose la fermeture des barrières si elles sont en position haute (horloge des instants t_2),
- **MONTER** qui indique la sortie des trains du secteur central (horloge des instants t').

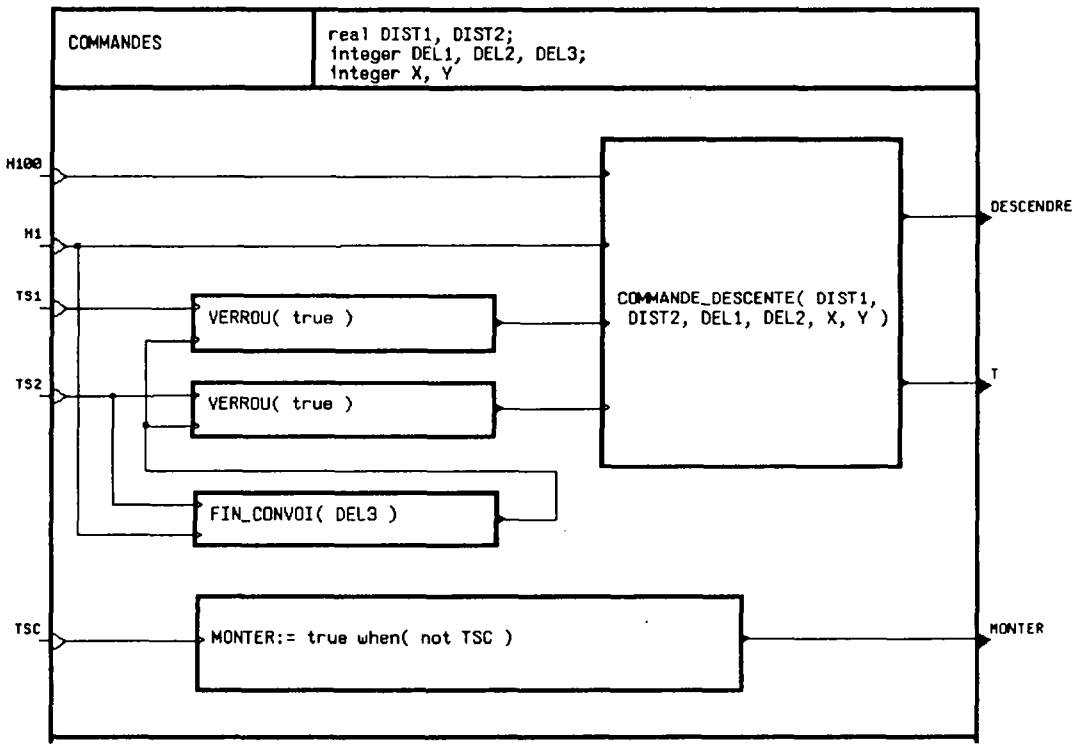


Figure III.6 : le processus **commandes**

Conformément au codage adopté pour exprimer l'occupation des voies (cf III.3), **MONTER** correspond à l'horloge des valeurs *faux* du signal **TSC** (figure III.6).

En faisant l'hypothèse que les trains ont une vitesse constante, les deux instants t_1 et t_2 peuvent être calculés en mesurant la durée du parcours entre les deux capteurs. Pour cela il faut donc détecter l'arrivée des trains au niveau de ces capteurs et la signaler au module **commande_descente** qui produit **T** et **DESCENDRE**.

La détection de l'arrivée des trains

Les capteurs produisent un top au passage de chaque wagon ; pour détecter qu'un train arrive au niveau d'un capteur, il faut donc reconnaître parmi ces tops ceux qui proviennent du premier wagon et inhiber les suivants jusqu'à la fin du train.

Ce filtrage est effectué sur les signaux **TS1** et **TS2** par deux processus du modèle **verrou** :

```
process VERROU =
      (logical INITIAL)
      { ? event RAZ, ENTREE
        ! event SORTIE}

      ( | PASSANT := RAZ default (not ENTREE)
        | ZPASSANT := PASSANT $1
        | SORTIE := ENTREE when ZPASSANT
        | )
where
      logical PASSANT, ZPASSANT init INITIAL
end
```

Ces **verrous** transmettent en **SORTIE** la première occurence de l'**ENTREE** (qui correspond ici à **TS1** ou **TS2**) qui suit une occurence du signal de réinitialisation **RAZ**.

Cette réinitialisation est effectuée, par le processus **fin_convoi**, lorsque la totalité du train a dépassé le deuxième capteur. Sans information sur le nombre de wagons, elle résulte de l'absence du signal **TS2**, pendant un délai **DEL3** supérieur à la durée maximale de passage d'un wagon.

L' ensemble **verrous**, **fin_convoi** fonctionne correctement si le délai minimal entre deux trains successifs est supérieur à **DEL3**. Les valeurs de vitesse maximale et de distance entre les trains, données dans les spécifications, assurent cette condition.

Le processus commande_descente

En connaissant le temps de parcours δ de chaque train entre les deux capteurs **TS1** et **TS2**, le système peut calculer la durée Δ que mettra ce train à atteindre le passage à niveau :

$$\Delta = \delta * \frac{\mathbf{DIST1}}{\mathbf{DIST2}}$$

Il lui faut attendre, après que le train a atteint **TS2**, l'écoulement des délais $(\Delta - \mathbf{DEL1} - \mathbf{DEL2})$ et $(\Delta - \mathbf{DEL1})$ pour produire respectivement les signaux **T** et **DESCENDRE**.

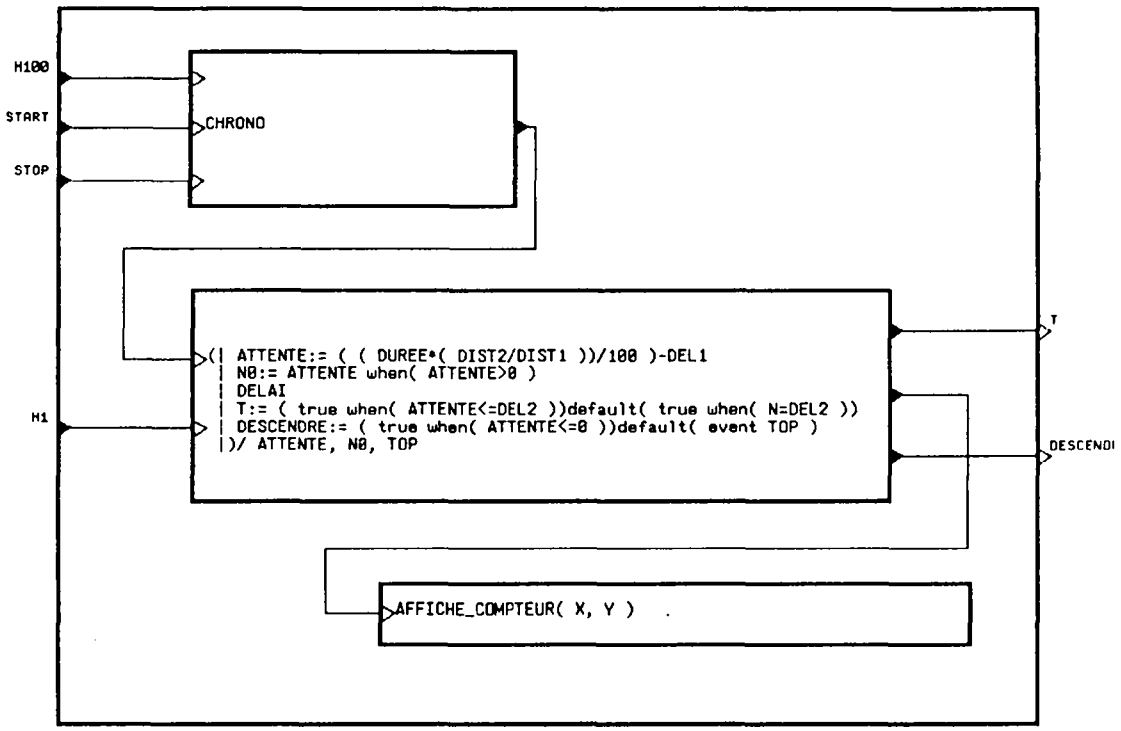


Figure III.7 : le processus **commande_descente**

Le processus **commande_descente** qui produit **T** et **DESCENDRE** comporte donc (figure III.7) :

- un chronomètre qui calcule la durée (δ) du parcours **TS1**–**TS2**,
- un module qui calcule et décompte un délai correspondant à Δ – **DEL1** pour produire **T** et **DESCENDRE**.

A ces deux modules vient s'ajouter une *sonde* : **affiche_compteur** (cf V.3) utilisée, dans l'interface graphique, pour afficher le décompte.

Le signal **START**, issu du verrou associé à **TS1** déclenche le **chrono**. La **DUREE** (δ) du parcours, mesurée en 1/100 s, est produite quand le signal **STOP** signale l'arrivée du train en **TS2**.

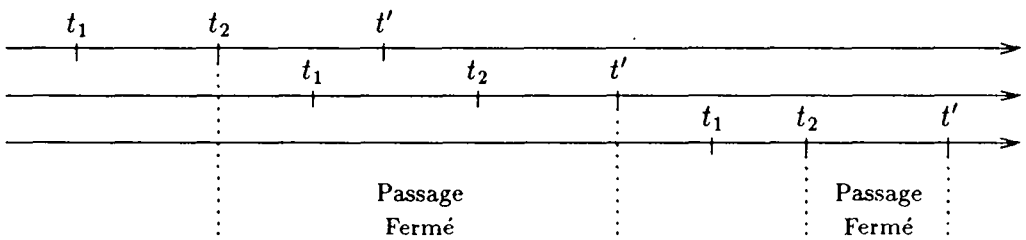
L' **ATTENTE** (Δ – **DEL1**), exprimée en secondes, est alors calculée, et initialise, si elle est positive, le **delai**, par l'intermédiaire du signal **N0**. La sortie **N** de ce dernier reflète le décompte, l'autre sortie **TOP** est produite quand **N** devient nul.

Le signal **T** est produit quand **N** est égal à **DEL2** ou, par précaution, simultanément à **ATTENTE** si cette dernière est inférieure à **DEL2**. De même, **DESCENDRE** correspond à l'émission de **TOP** ou à l'occurence d'une **ATTENTE** négative.

III.5.2 Le processus filtre_commandes

Considérons l'ensemble des trains circulant sur les deux voies ; comme on l'a vu précédemment, le parcours de chacun d'eux comporte deux phases successives qui déterminent la fermeture du passage :

- Une phase t_1–t_2, pendant laquelle l'ouverture des barrières est interdite.
- Une phase t_2–t', pendant laquelle la circulation doit être coupée.



A un instant t', un train quitte le secteur central, les barrières sont donc fermées. On doit les ouvrir, si aucun autre train n'est, au même instant, dans l'une des phases t_1–t_2 ou t_2–t'. Pour vérifier cette condition, il suffit de compter le nombre de trains situés à un instant de leur parcours postérieur à t_1 et antérieur à t'.

Une commande de fermeture doit être émise à un instant t_2, si les barrières sont ouvertes, c'est à dire au premier instant t_2 qui suit une commande d'ouverture.

Le processus **filtre_commandes** (figure III.8), chargé de produire les ordres **OUVRIR** et **FERMER**, reçoit les signaux $\mathbf{M}_i$, $\mathbf{T}_i$, $\mathbf{D}_i$ qui sont respectivement les signaux **MONTER**, **T** et **DESCENDRE** issus du processus **commandes** associé à la voie i.

Il est composé de deux modules, le premier **compte_trains** commande l'ouverture, le deuxième commande la fermeture.

Dans **compte_trains**, une occurence d'un signal $\mathbf{T}_i$, correspondant à un instant t_1, incrémente un compteur ; Une occurence de $\mathbf{M}_i$, correspondant à t', le décrémente. Le signal **OUVRIR** est l'horloge des valeurs nulles de ce

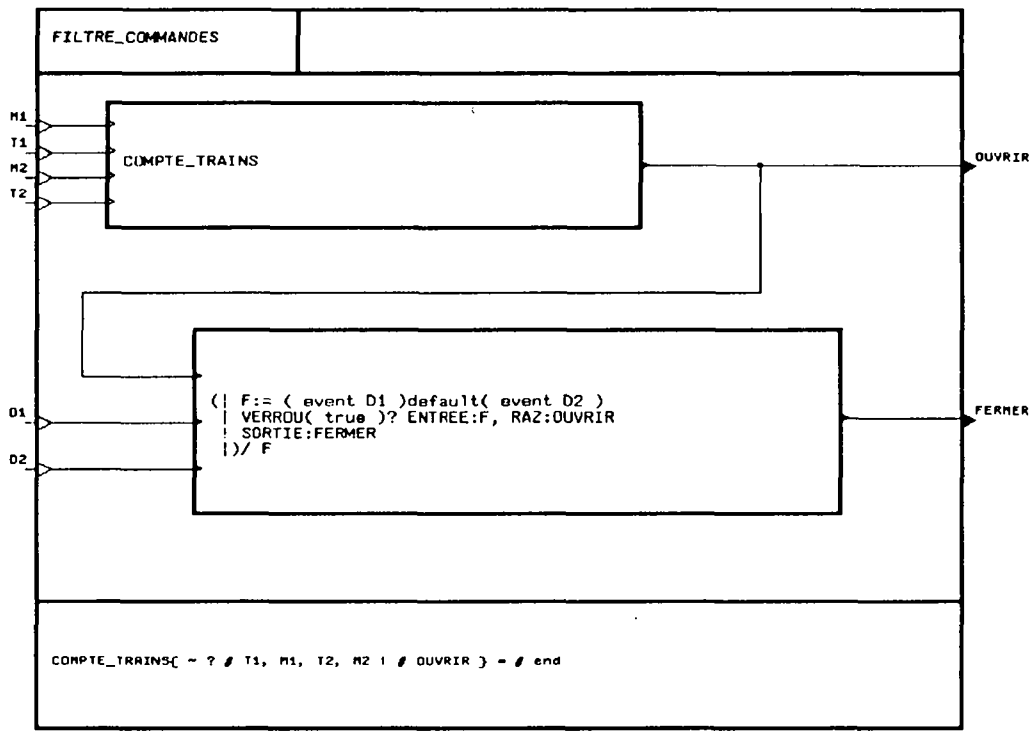


Figure III.8 : le processus **filtre_commandes**

compteur, c'est à dire l'ensemble des instants t' où la condition d'ouverture est satisfaite.

Le deuxième module extrait du signal **F**, horloge des instants t_2, ceux qui sont immédiatement postérieurs à une commande d'ouverture. Cette fonction est réalisée par une instance du modèle **verrou** déjà rencontré (cf III.5.1).

III.5.3 Le processus controle_barriere

Les deux barrières doivent, en théorie, être dans une position identique. Elles ne sont toutefois pas toujours dans le même état de fonctionnement. Il est donc nécessaire d' associer à chacune un contôleur indépendant.

Ces derniers sont chargés de transmettre les commandes d'ouverture et de fermeture aux barrières, d'en vérifier l'éxécution, et de signaler les pannes éventuelles au tableau de contrôle. Ils doivent, en outre, commander l'allumage et l'extinction des feux associés à chaque barrière.

Ils sont décrits par deux processus semblables, instances du modèle **controle_barriere** (figure III.9), chacun d'eux commandant un ensemble barrière-

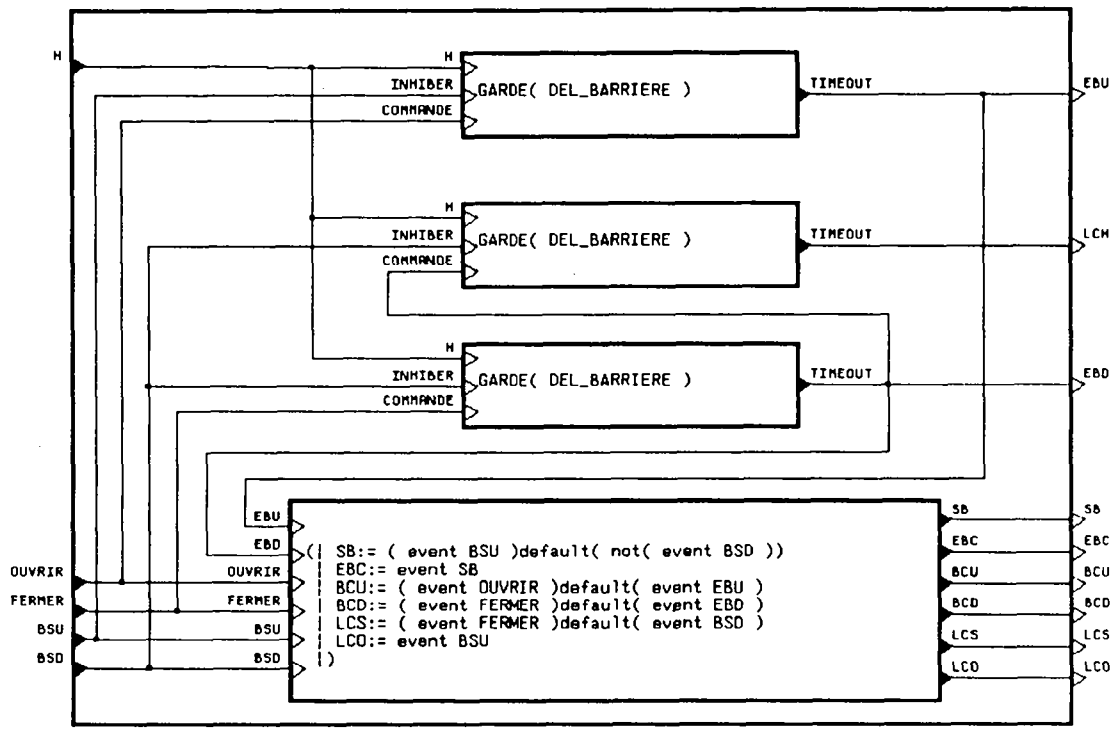


Figure III.9 : le processus **controle_barriere**

feu.

Ces deux processus mettent en œuvre les procédures d'ouverture et de fermeture respectivement déclenchées par l' occurence des signaux **OUVRIR** et **FERMER**.

Une occurence du signal **OUVRIR** provoque l'émission d'une commande d'ouverture et peut donner lieu, après un délai **DEL_BARRIERE**, à une deuxième commande, si le capteur **BSU** n'a pas indiqué l'ouverture effective.

De la même façon, **FERMER** produit une première commande de fermeture, suivie d'une deuxième, puis de l'allumage continu du feu si **BSD** ne signale pas la fermeture effective.

Il s'agit donc, une commande étant émise à un instant t, de signaler, à l'instant t + **DEL_BARRIERE**, l'échec de cette commande, si le capteur correspondant n'a pas réagi entre t et t + **DEL_BARRIERE**.

Cette fonction est assuré par chacun des processus du modèle **garde** ; l'un d'eux détecte l'échec de l'ouverture, les deux autres signalent respectivement l'échec de la première et de la deuxième commande de fermeture (figure III.9).

La détection de l'échec d'une commande

Le modèle **garde** produit le signal **TIMEOUT** dont les occurences indiquent que la **COMMANDE** a échoué. Pour cela, **COMMANDE** initialise un compteur à la valeur **DEL_BARRIERE**, décrémenté de manière synchrone à l'horloge **H**. L'occurence du signal **INHIBER** en provenance du capteur interrompt le décompte ; sinon celui-ci se poursuit et quand le compteur est nul, un top est émis sur la sortie **TIMEOUT**.

La transmission des commandes

Les diverses commandes destinées à la barrière, au feu ou au tableau de contrôle, sont déterminées aisément :

- **EBU** (panne en ouverture) : échec d'**OUVRIR**,
- **BCU** (commande d'ouverture) : **OUVRIR** ou échec de cette première commande (i.e. **EBU**),
- **EBD** (panne en fermeture) : échec de **FERMER**,
- **BCD** (commande de fermeture) : **FERMER** ou **EBD**,
- **LCH** (allumage continu) : échec de la deuxième commande de fermeture (i.e. de **EBD**),
- **LCO** (extinction du feu) : ouverture de la barrière.

Les changements de position de la barrière sont représentées par les valeurs *vrai* pour l'arrivée en position haute, *faux* pour la position basse du signal booléen **SB**.

Après deux échecs successifs en fermeture, le feu est allumé en continu. Si, par la suite, la panne est réparée, la barrière se ferme et on peut alors remettre le feu en position clignotante. C'est pourquoi le clignotement **LCS** est commandé lors de la première commande de fermeture **FERMER** et lors de l'arrivée en position basse de la barrière.

Chapitre IV

L'environnement de simulation

Pour en vérifier le comportement, le système de contrôle est testé dans un environnement de simulation qui autorise à tout moment l'intervention de l'utilisateur, tout en assurant que les contraintes imposées par les spécifications, à l'exception de la stabilité des vitesses, sont satisfaites.

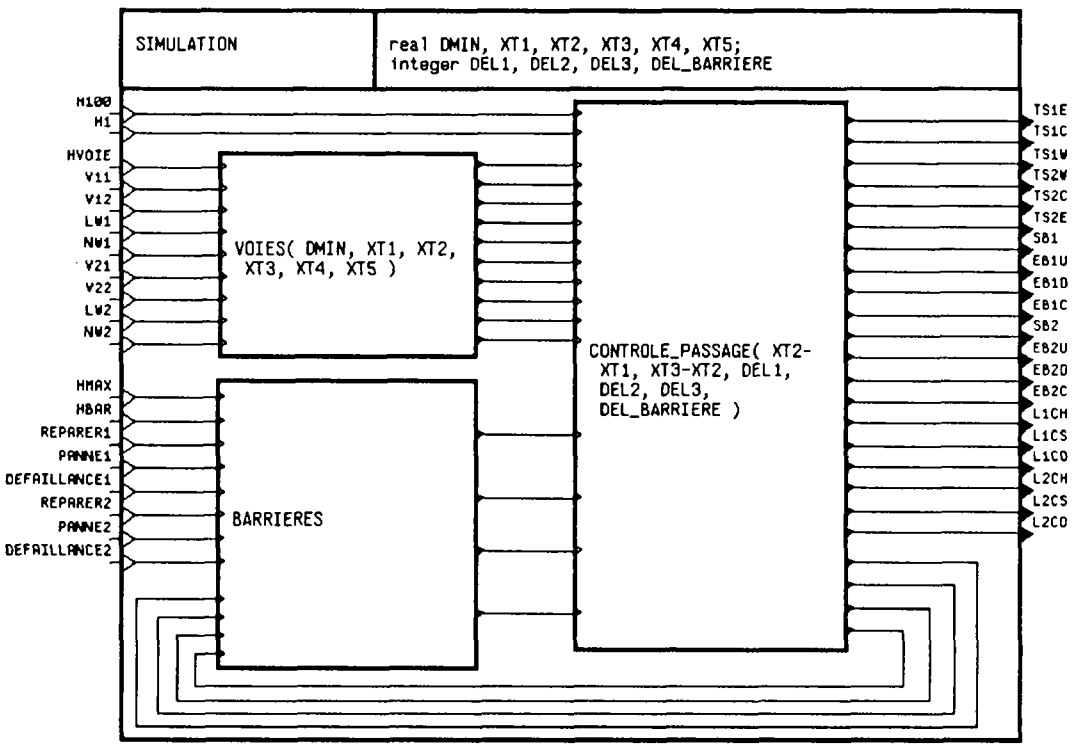


Figure IV.1 : le processus **simulation**

L'opérateur peut :

- lancer des trains, de longueur et nombre de wagons variables, sur chacune des voies,
- faire varier la vitesse de ces trains,
- introduire des défauts de fonctionnement au niveau des barrières et les réparer.

La possibilité de modifier à tout moment la vitesse des trains procure plus de facilités de simulations. Toutefois, pour maintenir le système de contrôle dans un état sain, il ne faut pas accélérer un train quand les barrières sont encore ouvertes.

Tous les paramètres de la simulation sont accessibles et modifiables à volonté, mais le simulateur n'effectue ces modifications que si elles ne contredisent pas une des contraintes (à l'exception de la stabilité des vitesses). On ne peut pas, par exemple, changer le nombre ou la longueur des wagons d'un train en circulation.

Le simulateur est naturellement décomposé en quatre entités indépendantes :

- Deux processus de simulation des voies identiques rassemblés dans **voies**.
- Deux processus de simulation des barrières rassemblés dans **barrières**.

Le processus **voies** reçoit les différents paramètres (vitesse, longueur et nombre des wagons) contrôlant la création et l'évolution des trains, ainsi qu'un signal d'horloge **HVOIE** qui en rythme les déplacements. Il simule les capteurs en produisant les signaux $\mathbf{T_iS_j}$ transmis au système de contrôle. Les paramètres de **voies** spécifient les positions des capteurs ($\mathbf{XT_1}$ à $\mathbf{XT_5}$) ainsi que la distance minimum entre deux trains successifs (**DMIN**).

Le processus **barrieres** utilise deux signaux d'horloge **HBAR** et **HMAX** et reçoit, outre les différentes actions de l'opérateur, les commandes d'ouverture et de fermeture émises par le contôleur. En retour, il transmet à celui-ci les signaux d'acquitement $\mathbf{B_iSU}$ et $\mathbf{B_iSD}$ quand les mouvements des barrières sont achevés.

IV.1 La simulation des voies

Pour le système de contrôle, chaque voie est perçue comme un groupe de cinq capteurs activés au passage des wagons. Le rôle de la simulation des voies est de produire les signaux issus de ces capteurs, correspondant aux trains que l'utilisateur a lancés.

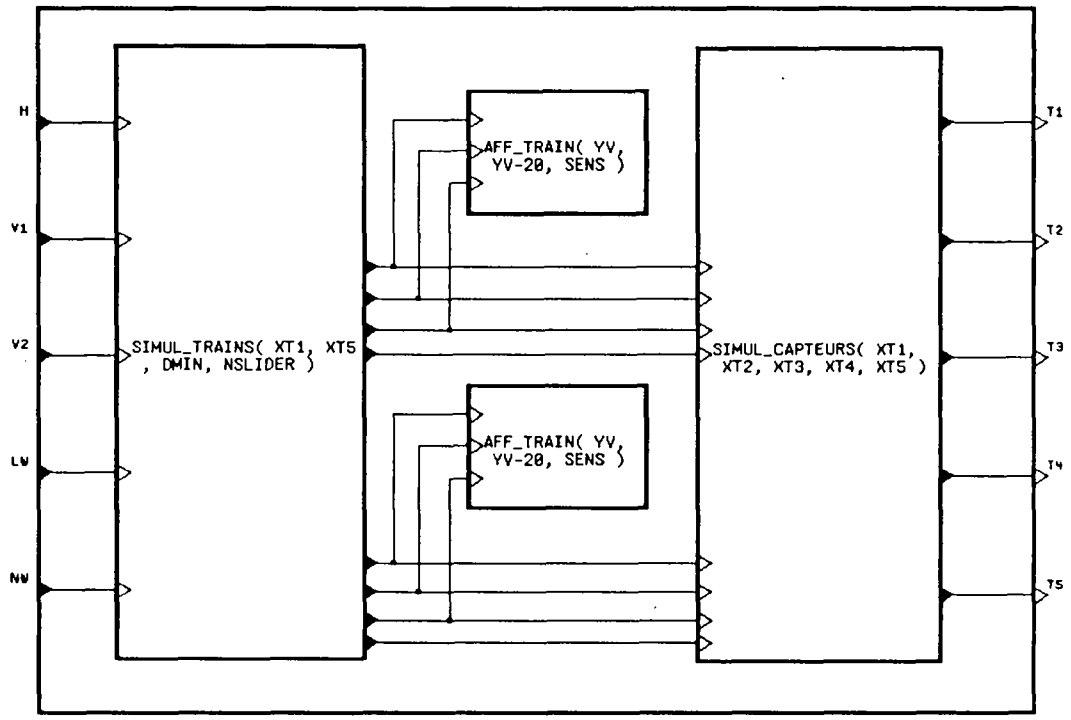


Figure IV.2 : le processus **simul_voie**

La simulation d'une voie comporte donc deux sous-ensembles :

- La simulation du parcours des trains
- La simulation des capteurs.

Le nombre de trains pouvant circuler simultanément sur une même voie est limitè ; la distance entre deux trains étant supérieure à 4000 métres, on a au maximum deux trains dans chaque sens. Quand l'un d'eux a achevé son parcours, il est reçyclé et devient disponible pour un nouveau parcours.

Chacun des processus **simul_voie** (figure IV.2) est décomposé en **simul_trains**, capable de faire circuler simultanément deux trains sur la voie, et **simul_capteurs** qui, à partir des coordonnées respectives de ces deux trains, produit les tops au passage de chaque wagon sur les capteurs.

On a choisi de représenter chaque train par l'ensemble des 4 signaux suivants :

- **X** suite des abscisses successives de la tête du train,
- **XX**, abscisses de la queue du train,

- **V**, vitesse du train,
- **LW**, longueur des wagons.

Ces signaux sont transmis au simulateur des capteurs, ainsi qu'à deux processus d'affichage **aff_train** qui n'interfèrent en rien avec la simulation (cf V.3).

L'utilisateur agit sur les trains par l'intermédiaire des signaux **V1** et **V2** qui fixent les vitesses, **LW** et **NW** qui définissent la longueur et le nombre de wagons. Les déplacements des trains sont cadencés par l'horloge **H**, une vitesse correspondant à la distance parcourue entre deux tops successsifs de **H**.

IV.1.1 La simulation des trains

Comme indiqué précédemment, simuler les trains consiste à émettre les signaux correspondant aux positions, vitesses et longueur de wagons successives, en respectant les contraintes suivantes :

- la longueur et le nombre des wagons sont fixés au début d'un parcours et restent constants tout au long de ce parcours.
- la distance entre les deux trains doit être supérieure à **DMIN**.
- les trains doivent être reçyclés en fin de parcours.

Un parcours débute quand, un train étant en position de départ **XDEB**, l'utilisateur lui communique une vitesse positive. A cet instant, la configuration de ce train (longueur et nombre de wagons) est déterminée. Le parcours se poursuit jusqu'à ce que le train dépasse l'abscisse **XFIN** de fin de la voie ; le train est alors stoppé et sa position réinitialisée à **XDEB**.

Quand deux trains circulent simultanément, la vitesse du premier est libre, le second doit ajuster sa vitesse pour ne pas enfreindre la condition de distance minimale. Pour mettre en œuvre simplement ce contrôle, on a choisi d'arrêter momentanément le deuxième train tant que la distance est inférieure à **DMIN**.

La simulation des deux trains est décrite par **simul_trains** (figure IV.3). Ce dernier comporte un contrôleur (**contrôle**) qui détermine, à partir des vitesses, longueurs et nombres de wagons communiqués par l'utilisateur, la configuration et la vitesse effective de chacun des trains. Ces informations sont transmises à deux instances du modèle **train** qui calculent les positions successives des deux trains.

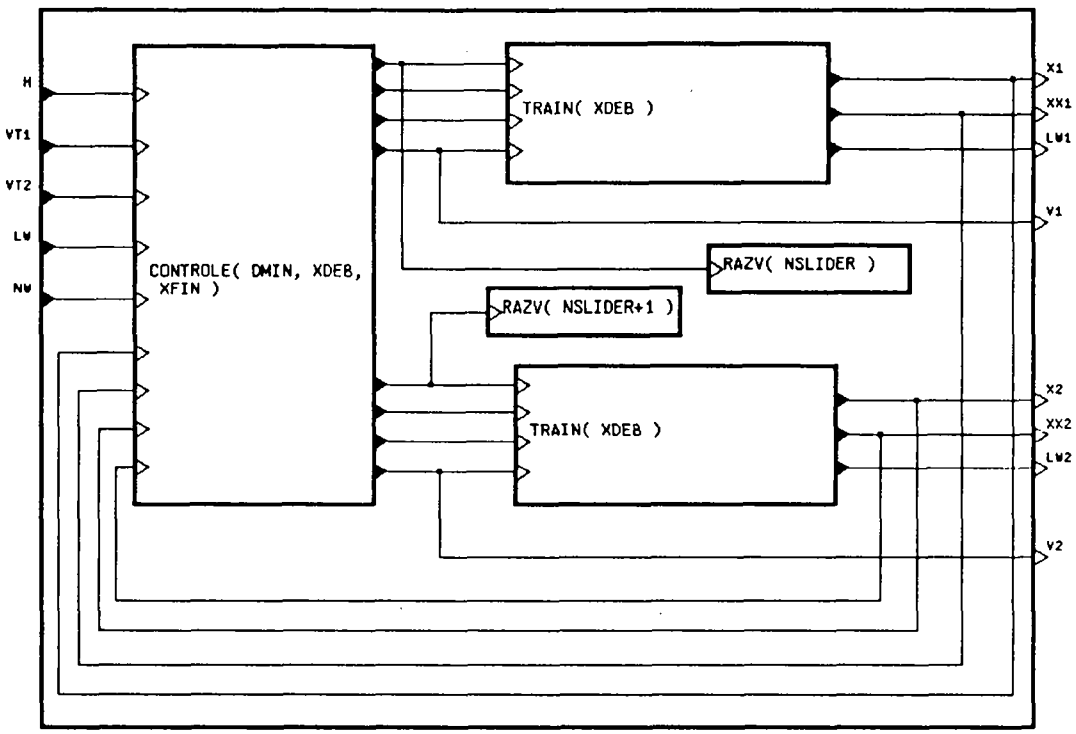


Figure IV.3 : le processus **simul_trains**

Le processus de contrôle

Ce processus est chargé de déteminer, lors du démarrage d'un train, le nombre et la longueur des wagons, de transmettre la vitesse demandée par l'utilisateur quand celle-ci respecte la condition de distance minimale, de stopper le train quand son parcours est achevé.

Les deux trains ont des comportements symétriques ; ils peuvent être indiféremment en première ou en deuxième position sur la voie. Ils respectent donc la même discipline et sont contrôlés par deux processus **controle_vitesse** identiques :

```
process CONTROLE_VITESSE  =
     (real DMIN, XDEB, XFIN)
     { ? event H;
         real ZX, ZXAUTRE, ZXX, ZXXAUTRE, VT, LW;
         integer NW
       ! real V, LWAG;
         integer NWAG;
         event STOP}

     ( | STOP   := when (ZXX > XFIN)
       | VITESSE  := (0.0 when STOP) default VT
           default ZVITESSE
       | ZVITESSE  := VITESSE $1
       | synchro {VITESSE, H}
       | HV   := STOP default (when ((VITESSE > 0.0)
           and ((ZX >= ZXAUTRE) or (ZX+DMIN < ZXXAUTRE))))
       | V   := VITESSE when HV
       | DEMARRAGE  := (event V) when (ZX = XDEB)
       | L   := LW cell DEMARRAGE
       | LWAG   := L when DEMARRAGE
       | N   := NW cell DEMARRAGE
       | NWAG   := N when DEMARRAGE
       | )
where
     real VITESSE, ZVITESSE init 0.0;
     event DEMARRAGE, STOP, HV;
     real L init 10.0;
     integer N init 10
end
```

ZX et **ZXX** sont les abscisses du train contrôlé, **ZXAUTRE** et **ZXXAUTRE**, celles de l'autre train.

Le signal **STOP** détecte la fin d'un parcours.

La **VITESSE** théorique devient nulle après un parcours, autrement, elle est égale à la dernière vitesse **VT** communiquée par l'utilisateur. **VITESSE** est synchrone à **H** qui rythme donc le déplacement du train.

Cette **VITESSE** théorique est transmise en **V** à l'occurence du signal **STOP** ou quand elle est positive et que la condition de distance est respectée.

Un **DEMARRAGE** est détecté quand le train est en position de départ

XDEB et qu'une vitesse **V** est transmise. La longueur et le nombre de wagons, mémorisées par **L** et **N** sont alors émis en **LWAG** et **NWAG**.

Pour une synchronisation correcte, il faut que les signaux **ZX**, **ZXX**, **ZX-AUTRE** et **ZXXAUTRE** soient synchrones à **H**. Ils sont produits en suréchantillonnant les signaux de position des trains comme le décrit le modèle **resync** :

```
process RESYNC =
     { ? real X;
         event H
       ! real ZX}

     (| Z  := X default ZX
      | ZX  := Z $1
      | synchro {Z, H}
      |)
where
     real Z
end
```

Deux autres contrainte de synchronisation doivent être satisfaites. La première, c'est que le signal **VT** soit moins fréquent que **H**. Ceci est assuré par le module de scrutation des entrées (cf V.1.2). La deuxième, c'est que les deux trains ne démarrent pas simultanément. Il est possible de le contrôler en SIGNAL, par exemple, en donnant la priorité à l'un des trains. Mais, en pratique, cette condition est évidemment respectée.

Le modèle train

Les parcours successifs d'un train sont simulés par un processus **train** qui calcule les positions **X** et **XX** de début et de fin du train, en fonction de la vitesse **V**, de la longueur et du nombre de wagons et du signal de fin de parcours **STOP**. En outre, il émet, de manière synchrone à **V**, la longueur des wagons.

```
process TRAIN =
        (real XDEB)
      { ? real V, LWAG;
          integer NWAG;
          event STOP
        ! real X, XX, LWAGONS}

      (| X   := (XDEB when STOP) default (ZX + V)
       | XX   := X - LTRAIN
       | ZX   := X $1
       | LTRAIN   := (LWAG * NWAG) default ZLTRAIN
       | ZLTRAIN   := LTRAIN $1
       | LWAGONS   := LWAG default ZLWAGONS
       | ZLWAGONS   := LWAGONS $1
       | synchro {LWAGONS, V}
       |)

where
      real ZX init XDEB, LTRAIN, ZLTRAIN init 100.0,
           ZWAGONS init 10.0
end
```

Un train est ainsi représenté par quatre signaux synchrones : **X**, **XX** , **LWAGONS** produits par **train**, et **V** produit par le contrôleur.

IV.1.2 Simulation des capteurs

Chaque voie comporte un ensemble de cinq capteurs, chacun d'entre eux devant produire un top lors du passage d'un wagon sur la portion de voie qu'il occupe. Les cinq capteurs sont simulés par des processus identiques, qui reçoivent les mêmes signaux (abscisses, vitesse et longueur des wagons) et ne diffèrent que par l'emplacement du capteur qu'ils représentent. Ces processus sont rassemblés dans **simul_capteurs**, les paramètres **XT1** à **XT5** déterminent leurs positions respectives le long de la voie.

Chacun de ces processus est une instance du modèle **mux_capteur** qui est formé de deux processus **activite_capteur**, affectés chacun à la surveillance d'un des deux trains qui circulent sur la voie, et d'un **capteur** proprement dit.

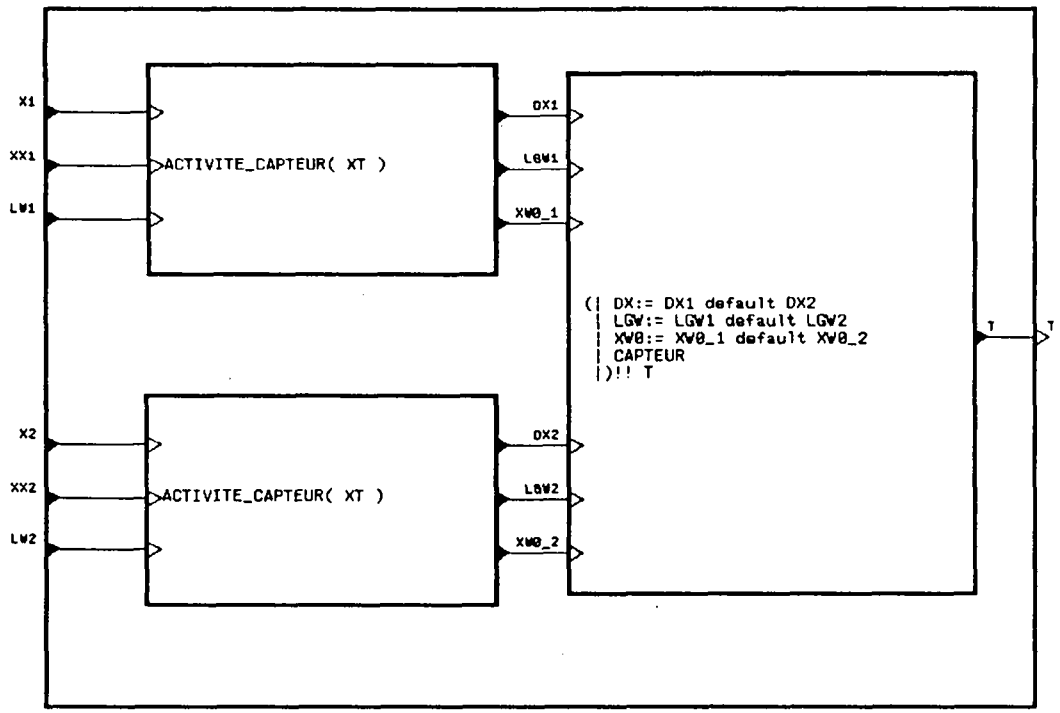


Figure IV.4 : simulation d'un capteur

Les processus de surveillance

Les processus **activite_capteur** surveillent chacun un train. Ils alertent le **capteur** quand ce train atteint la position **XT** puis lui transmettent, tant que la totalité du train n'a pas dépassé **XT**, la vitesse du train et la longueur des wagons.

Un processus **activite_capteur** est donc actif tant que $\mathbf{XX} \leq \mathbf{XT} < \mathbf{X}$, les signaux **X**–**XX** représentant les coordonnées du train.

Durant cette phase d'activité, la longueur **LGW** des wagons et la vitesse **VT** du train sont émis. La valeur **XW0** d'initialisation du **capteur** est générée lors de l'activation (quand **ACTIF** devient *vrai* alors que **ZACTIF** est *faux*).

```
process ACTIVITE_CAPTEUR =
     (real XT)
     { ? real X, XX, V, LW
       ! real VT, LGW, XW0}

     ( | ACTIF := (X > XT) and (XX <= XT)
       | ZACTIF := ACTIF $1
       | XW0 := (X - XT) when (ACTIF and (not ZACTIF))
       | LGW := LW when ZACTIF
       | VT  := V when ZACTIF
       | )
where
     logical ACTIF, ZACTIF init false
end
```

Puisque deux trains ne peuvent pas être ensemble au niveau du même capteur, un seul processus de simulation du capteur suffit. Il faut simplement l'alimenter par les fusions respectives des **XW0**, **VT** et **LGW** provenant de chaque processus de surveillance.

Le capteur

On suppose, pour la simulation, que le top signalant un wagon est produit quand la fin de ce wagon dépasse le capteur. Il faut donc déterminer, quand un wagon est au niveau d'un capteur, la distance entre la fin de ce wagon et ce capteur.

Considérons un train comportant des wagons de longueur LGW et un capteur situé à la position XT sur la voie. Appelons $XW(n)$ la distance entre le n^{ieme} wagon et le capteur, et VT la vitesse du train.

A tout instant t, les équations suivantes sont vérifiées :

$$\begin{aligned} XW(n)_t &= XW(n)_{t-1} + VT_t \\ XW(n)_t &= XW(n-1)_t - LGW \end{aligned}$$

Le top correspondant au n^{ieme} wagon est produit au premier instant t tel que $XW(n)_t \geq LGW$.

Ces équations peuvent être aisément traduites en SIGNAL, mais nécessitent de gérer un signal pour chaque wagon (ou un vecteur comportant autant de composantes qu'il y a de wagons). Ce qu'on désire plutôt c'est un signal qui indique à tout instant t, la distance entre **XT** et le wagon qui à cet instant est au dessus du capteur. C'est à dire le signal XW défini par : $XW_t = XW(n)_t$ ou n est l'unique indice tel que $0 \leq XW(n)_t < LGW$.

Si $XW_t = XW(n)_t$ deux cas sont possibles pour l'instant $t-1$:

(1) $XW_{t-1} = XW(n)_{t-1}$, on en déduit que

$$XW_t = XW_{t-1} + VT_t$$

(2) $XW_{t-1} = XW(n-1)_{t-1}$ ce qui signifie que $XW(n-1)_t \geq LGW$; un top est produit à l'instant t et

$$\begin{aligned} XW_{t-1} + VT_t &\geq LGW \\ XW_t &= XW_{t-1} + VT_t - LGW \end{aligned}$$

On peut donc, grâce à ces équations, définir XW sans calculer explicitement chaque $XW(n)$:

```
( | NXW  := XW $1 + V
  | XW   := ((NXW - LGW) when (NXW >= LGW))
                default NXW
  | )
```

Pour tenir compte du passage des trains successifs, il faut réinitialiser **XW** chaque fois qu'un nouveau train dépasse le capteur. Cette réinitialisation est effectuée par le signal **XW0**, distance entre la tête d'un train et le capteur, reçu quand ce train atteint **XT**.

Voici la définition complète du processus **capteur** :

```
process CAPTEUR =
     { ? real VT, LGW, XW0
       ! event T}

     ( | Z := ZXW when (event VT)
       | NXW := Z + VT
       | T := when (NXW >= LGW)
       | XW := XW0 default ((NXW - LGW) when T)
           default NXW
       | ZXW := XW $1
       | )
where
     real Z, NXW, XW, ZXW init 0.e 0
end
```

IV.2 La simulation des barrières

En fonctionnement normal, une barrière émet, en réponse à une commande d'ouverture ou de fermeture, un signal d'acquittement, dans un délai inférieur à 5 secondes. Une barrière n'est pas fiable, elle peut être sujet à des pannes qui doivent être détectées par le système de contrôle. Pour vérifier le comportement de ce dernier, par la simulation, il est intéressant de fournir à l'opérateur les moyens de provoquer et de réparer des pannes de barrière.

La simulation d'une barrière comporte ainsi deux fonctions :

- simuler les mouvements des barrières physiques,
- simuler les pannes de ces barrières

L'opérateur a le contrôle du fonctionnement des barrières par l'intermédiaire, pour chacune d'entre elle, de trois commandes : **PANNE**, **DEFAILLANCE** et **REPARER**. Après une **PANNE**, la barrière n'obéit plus aux commandes d'ouverture et de fermeture. Une **DEFAILLANCE** est une panne passagère : la barrière ne réagit pas à la commande suivante mais retrouve, après cette dernière, un état correct. **REPARER** permet de rétablir le fonctionnement normal de la barrière.

Si une **PANNE** ou une **DEFAILLANCE** interviennent quand la barrière est en mouvement, celui-ci est interrompu. A l'inverse, **REPARER** peut provoquer le déplacement de la barrière, afin de la placer dans la position conforme à la dernière commande reçue.

Simuler pannes et réparations, consite donc à contrôler le mouvement des barrières en répercutant ou non les commandes d'ouverture et de fermeture vers un dispositif qui effectue les déplacement. Le processus de simulation d'une barrière est ainsi constitué (cf figure IV.5) :

- d'un automate de **controle**,
- d'un **moteur** qui simule les mouvements.

Deux signaux booléens, **MOUVEMENT** et **SENS**, disponibles à l'horloge **HMAX** sont élaborés par l'automate. Les valeurs *vrai* de **MOUVEMENT** réclament le déplacement de la barrière ; **SENS** en indique le sens (ouverture pour *vrai*, fermeture pour *faux*).

Le **moteur** effectue ces déplacements au rythme de l'horloge **HBAR**, et produit, quand ils sont achevés, les signaux d'acquittement **BSU** et **BSD**.

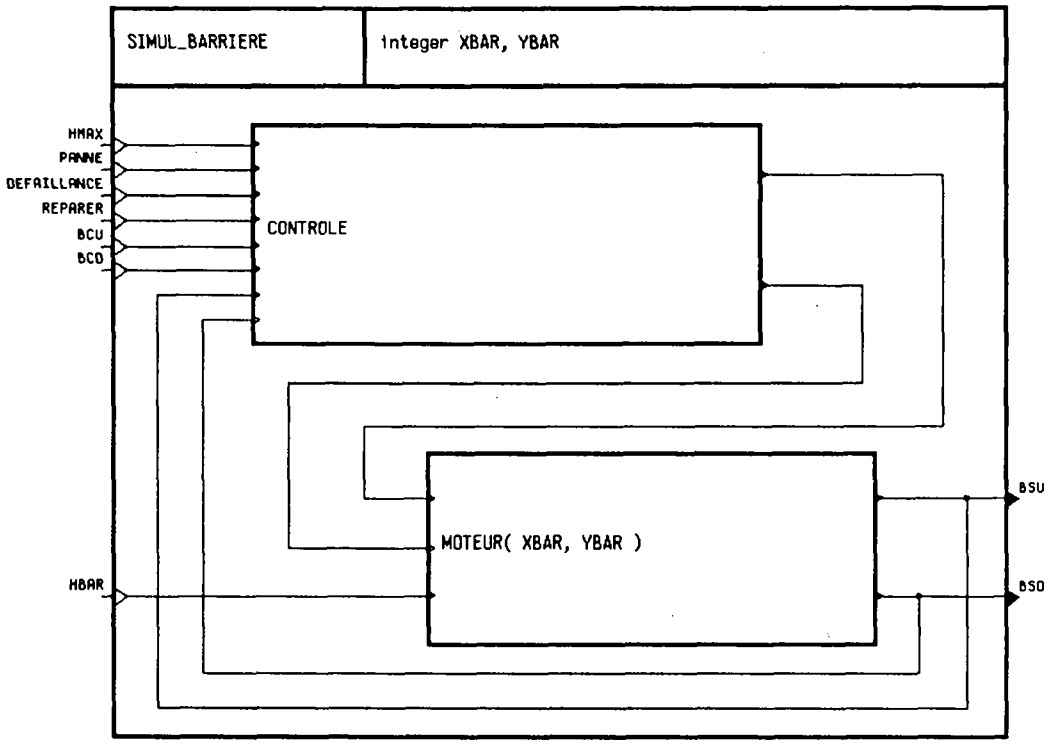


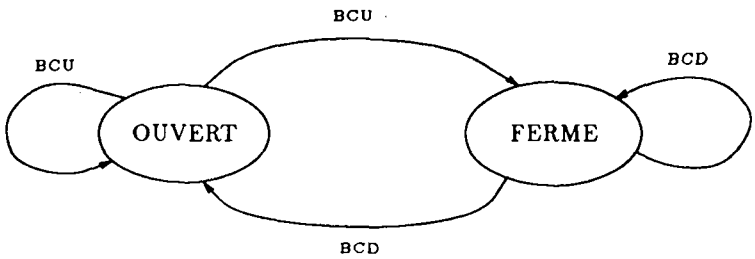
Figure IV.5 : simulation d'une barrière

Au **moteur** est associé un processus graphique. Les paramètres **XBAR** et **YBAR**, coordonnées de l'image de la barrière sont destinées à ce processus d'affichage (cf chapitre V.2).

IV.2.1 Le contrôle des mouvements de la barrière

Contrôler le mouvement des barrières c'est déterminer à quels instants elle doit se déplacer et dans quel sens.

Le sens de déplacement, est régi par les commandes d'ouverture (**BCU**) et de fermeture (**BCD**). Il correspond à la position théorique des barrières comme le décrit l'automate suivant :



Cet automate est celui décrit par le modèle **etat** (cf II.3.5). **SENS** est donc défini par :

```
SENS   := ETAT() {BCU, BCD, HMAX}
```

Où **HMAX** est l'horloge de consultation de l'**etat**.

Un mouvement est amorcé à la réception d'une commande, si le fonctionnement est normal, ou lors d'une réparation de la barrière. Il est interrompu par une **PANNE** ou une **DEFAILLANCE** ou par l'un des signaux d'acquittement **BSU** ou **BSD**. Si on appelle **ZPANNE** et **ZDEFAILLANT** deux signaux booléens qui indiquent repsectivement si la barrière est en panne ou en défaillance, on peut définir **MOUVEMENT** par :

```
( | COMMANDE := BCU default BCD
  | DEB_MV := REPARER default
      (COMMANDE when (not (ZPANNE or ZDEFAILLANT)))
  | FIN_MV := PANNE default DEFAILLANCE
      default BSU default BSD
  | MOUVEMENT  := ETAT() {DEB_MV, FIN_MV, HMAX}
  | )
```

Il reste reste à calculer les deux signaux **ZPANNE** et **ZDEFAILLANT**. On peut là encore utiliser le modèle **etat** et écrire :

```
( | FIN_DEF := REPARER default COMMANDE
  | ZPANNE  := ETAT() {PANNE, REPARER, HMAX}
  | ZDEFAILLANT  :=
      ETAT() {DEFAILLANCE, FIN_DEF , HMAX}
  | )
```

La fin d'une défaillance (**FIN_DEF**) correspond à un occurence de **REPARER** ou d'une **COMMANDE**. Conformément à la définition d'**etat**, **ZDEFAILLANT** devient faux à l'instant qui suit **FIN_DEF**. Ainsi, si une **COMMANDE** intervient, alors que **ZDEFAILLANT** est vrai, elle n'active pas

MOUVEMENT mais provoque la fin de la défaillance. La **COMMANDE** suivante pourra donc être répercutée et amorcera le **MOUVEMENT** comme le décrit le chronogramme suivant :

HMAX :	—	—	—	—	—	—	—	—	....
DEFAILLANCE :		—							....
ZDEFAILLANT :	*F*	*F*	*V*	*V*	*F*	*F*	*F*	*F*	....
COMMANDE :				—		—			....
DEB_MV :						—			....
MOUVEMENT :	*F*	*F*	*F*	*F*	*F*	*F*	*V*	*V*	....

IV.2.2 Le moteur de la barrière

Pour simuler les déplacements de la barrière, on a représenté sa position par un entier de 0 (barrière fermée) à 9 (barrière ouverte). Cette position est augmentée ou diminuée, suivant la valeur des signaux **POS_THEORIQUE** et **MOUVEMENT**, de manière synchone à **HBAR**. **BSU** est produit quand la position devient égale à 9, **BSD** quand elle devient nulle.

```
process MOTEUR =
     (integer XBAR, YBAR)
     { ? event HBAR;
         logical MOUVEMENT, POS_THEORIQUE
       ! event BSU, BSD}

     ( | DP := (((1 when POS_THEORIQUE) default
            (-1 when (not POS_THEORIQUE)))
            when MOUVEMENT) when HBAR
       | NP := ZP+DP
       | BSU := when (NP >= 9)
       | BSD := when (NP <= 0)
       | P := (9 when BSU) default (0 when BSD) default NP
       | ZP := P $1
       | DEPLACE_BARRIERE (XBAR, YBAR)  {P, ZP}
       | )
where
     integer DP, NP, P, ZP init 9
end
```

Le processus **deplace_barriere** dessine la barrière dans sa nouvelle position. Il fait appel pour cela à une fonction externe (cf V.3).

Chapitre V

L'interface graphique

Afin de permettre à l'opérateur de visualiser et de contrôler la simulation, cette dernière est munie d'une interface avec un système de fenêtrage (ici SUNVIEW). L'utilisateur peut agir sur l'ensemble des signaux de contrôle de la simulation (vitesse des trains, pannes des barrières ...) par l'intermédiaire de curseurs, boutons ...(cf. figure V.1). L'interface assure de plus l'affichage des trains, des barrières, des feux et du tableau de contrôle.

Elle est réalisée en construisant, autour du processus **simulation**, un processus de gestion des entrées qui consomme les événements qui se produisent dans la fenêtre et un processus **graphiques** qui récupère les sorties de **simulation** pour mettre en œuvre l'affichage (cf figure V.2). En outre, certaines opérations graphiques, comme l'affichage des trains et des barrières, utilisent des informations qui ne sont pas directement accessibles en sortie de **simulation**. Elles sont effectuées par des processus qui prélèvent ces informations sans interférer avec le reste de la simulation : les *sondes*.

V.1 La gestion des entrées

Les signaux d'entrée du processus de simulation se répartissent en deux catégories :

- d'une part, l'ensemble des horloges qui rythme les phénomènes simulés,
- d'autre part, les signaux qui permettent à l'utilisateur de contrôler ces phénomènes.

La gestion des entrées regroupe donc deux tâches :

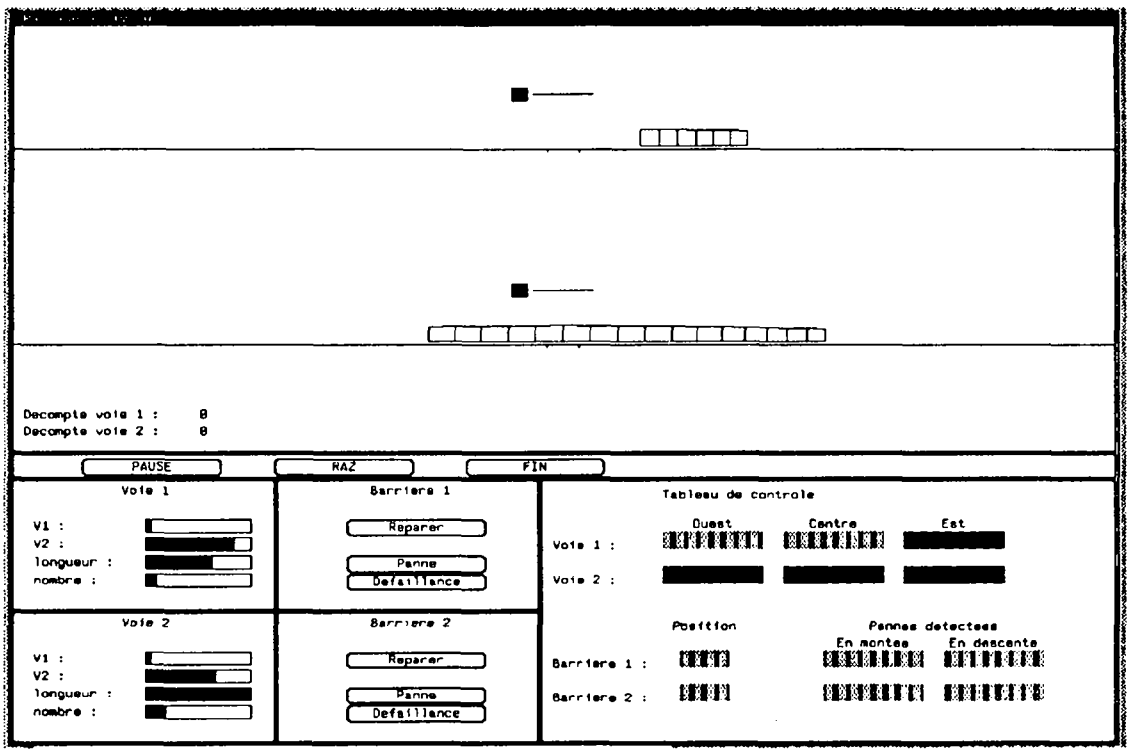


Figure V.1 : L'interface graphique

- produire, à partir d 'une horloge externe, les horloges de la simulation,
- transmettre quand l'opérateur agit sur l'un des dispositifs de commande, la valeur ad hoc sur le signal d'entrée associé.

V.1.1 La génération des horloges

A partir de l'horloge externe **H** sont construites les horloges suivantes :

- Les horloges **H100** et **H1** du système de contrôle,
- Les horloges **HVOIE**, **HBAR** et **HMAX** de la simulation,
- L'horloge **HGRAPH** utilisée par le processus **graphiques**.

Il existe des contraintes de synchronisation entre ces diverses horloges :

- **HMAX** doit être supérieure à **H100** et **HBAR**,

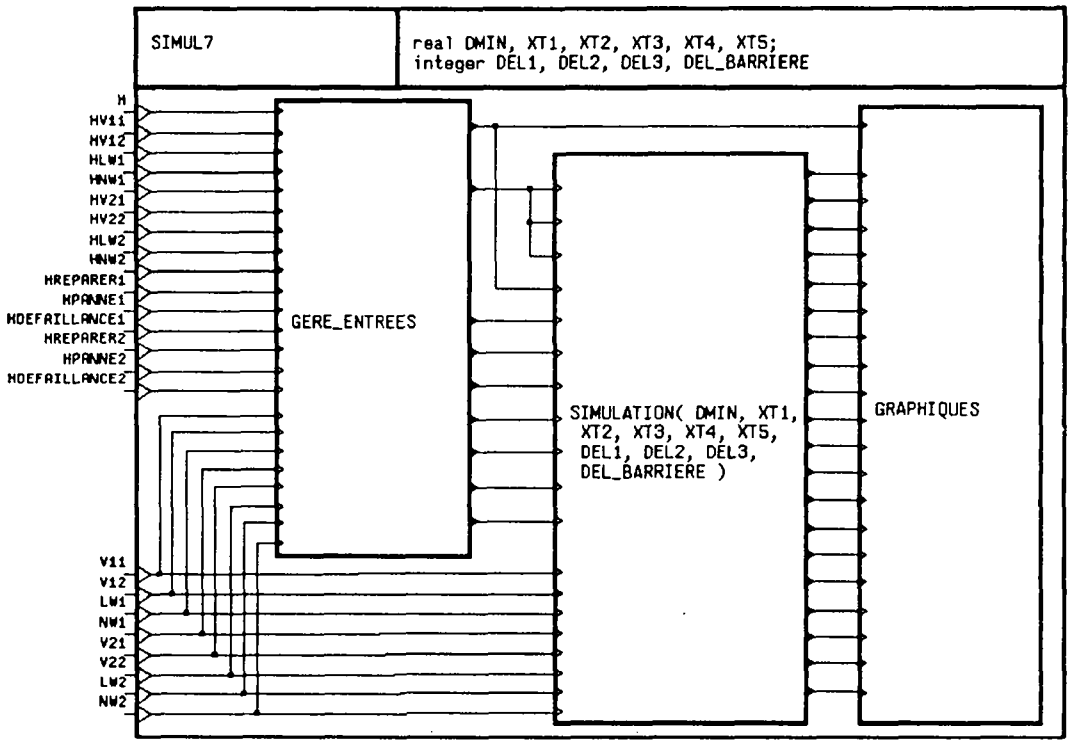


Figure V.2 : Le processus **simul**

- la fréquence de **H1** doit être 100 fois plus faible que celle de **H100**,

Il faut de plus cadencer convenablement **HBAR**, pour que le temps de réponse des barrières respecte les spécifications, et **HVOIE** pour que les mouvements des trains soient suffisamment précis.

On a choisi les définitions suivantes, qui assurent que toutes ces conditions sont satisfaites :

```
( | H100   := H
  | HVOIE  := H
  | HMAX   := H
  | HBAR   := MODULO(40) {H}
  | HGRAPH  := HBAR
  | H1   := MODULO(100) {H}
  | )
```

Le modèle **modulo**, dont la définition est donnée ci-dessous, décrit le sous-échantillonnage à fréquence fixe d'une horloge.

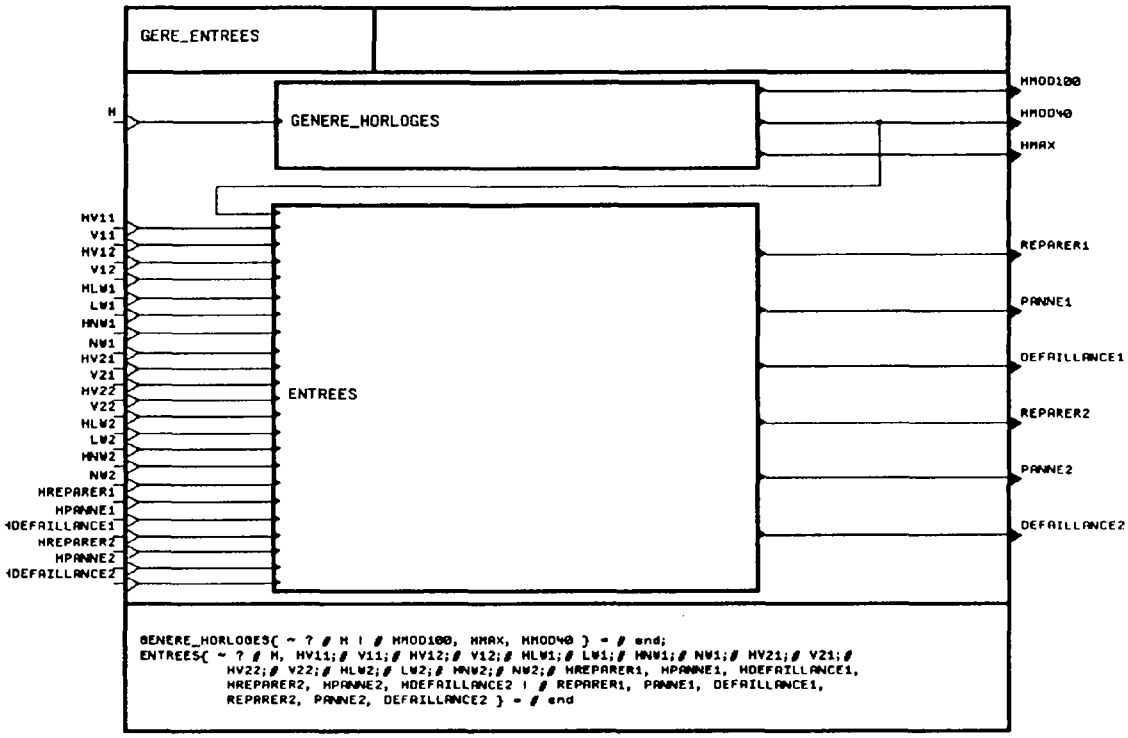


Figure V.3 : Le processus de gestion des signaux d'entrée

```
process MODULO =
      (integer M)
      { ? event HE
        ! event HS}

      (| N := (M when HS) default (ZN - 1)
       | ZN := N $1
       | synchro {N, HE}
       | HS := when (ZN = 1)
       |)
where
      integer N, ZN init 1
end
```

Un top de l'horloge de sortie **HS** est produit tous les **M** tops de l'horloge d'entrée **HE**.

V.1.2 La scrutation des entrées

L'opérateur dirige la simulation par l'intermédaire d'un groupe de signaux qui commandent chacun l'une des entités simulées (trains et barrières). Un objet graphique permet d'agir sur chacun de ces signaux (curseur pour les signaux numériques, bouton pour les commandes). Un signal booléen indique si un événement relatif à cet objet s'est produit ; l'entrée associée est donc synchrone aux valeurs vraies de ce signal.

A titre d'exemple, voici le modèle qui décrit la synchronisation des signaux de contrôle d'une voie :

```
process ENTREES_VOIE =
     { ? event H;
         logical HV1;
         real V1;
         logical HV2;
         real V2;
         logical HLW;
         real LW;
         logical HNW;
         integer NW
       ! }

     (| synchro {HV1, HV2, HLW, HNW, H}
      | synchro {V1, when HV1}
      | synchro {V2, when HV2}
      | synchro {LW, when HLW}
      | synchro {NW, when HNW}
      |)
end
```

V.2 Le processus d'affichage

Le processus d'affichage visualise le tableau de contôle, et les deux feux commandés par le processus **controle_passage**. L'occupation de chaque secteur, les positions et pannes des barrières, l'état des feux sont représentés par des **temoins**. Ces processus effectuent l'affichage en appelant une procédure externe.

```
process TEMOIN =
     (integer F, X1, Y1, X2, Y2)
     { ? logical T
       ! }

     ( | COULEUR := (1 when T) default (0 when (event T))
       | RECTANGLE() {F, X1, Y1, X2, Y2, COULEUR}
       | )
where
     integer COULEUR

     function RECTANGLE =
          { ? integer FEN, X1, Y1, X2, Y2, COULEUR
            ! }

end
```

Un événement de l'entrée **T** provoque l'affichage d'un **rectangle** de coordonnées données en paramètres, dans la **COULEUR** indiquée par **T**.

V.3 Les sondes

Pour extraire des valeurs internes au simulateur, on utilise des **sondes**, c'est à dire des processus qui, greffés à l'intérieur des divers modules de la simulation, se contentent de prélever certains signaux et d'effectuer l'action qu'ils déterminent, sans perturber le comportement du module.

Ce sont par exemple, les processus **aff_train** et **deplace_barriere**, déjà rencontrés, qui dessinent les trains et les barrières. Un autre exemple est la sonde **affiche_compteur** (figure V.4) qui permet de visualiser le compte à rebours qui précède la fermeture des barrières (cf figure V.4).

Ces sondes agissent de la même façon que les **temoins** :elles transmettent les signaux prélevés à une procédure externe qui effectue l'affichage.

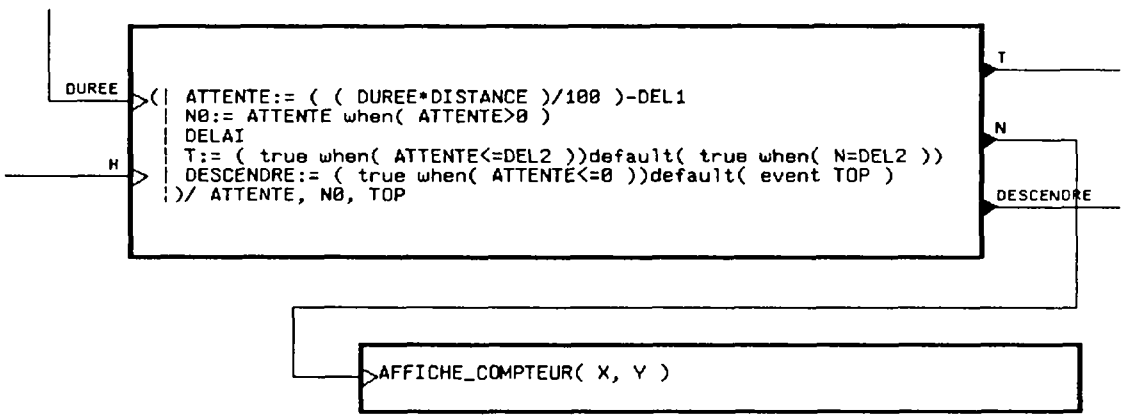


Figure V.4 : La sonde **affiche_compteur**

Chapitre VI

Conclusion

L'objectif fixé ici est d'illustrer une méthodologie de développement de systèmes temps réel en utilisant le langage SIGNAL. Le développement peut passer par plusieurs phases, et en particulier nécessiter la simulation, comme dans le cas présent. Ce que l'on souhaite c'est obtenir des outils permettant d'assurer que l'on met bien en œuvre ce que l'on a étudié.

On s'est donc attaché à décrire le programme temps-réel synchrone de gestion du passage à niveau qui pourrait fonctionner sur site, à l'intégrer dans une interface asynchrone/synchrone décrite en SIGNAL, à plonger le tout dans un environnement de simulation interactif.

L'approche descriptive permet une traduction directe du cahier des charges en une spécification SIGNAL complète. L'utilisation de systèmes d'équations permet une description modulaire très souple, mise en évidence dans l'éditeur graphique de construction de programme. Chaque module est défini uniquement en fonction des propriétés de synchronisation de ses entrées/sorties ; le comportement global du programme est synthétisé par le compilateur.

Pour que l'objectif soit complètement atteint, il faudrait, outre des générateurs de code pour les processeurs ou automates matériels utilisés, que les sondes servant à la visualisation puissent être gérées en arrière plan du programme par l'éditeur graphique. On obtiendrait alors un seul et même source pouvant fonctionner en simulation, en mise au point, et dans l'environnement réel.

Bibliographie

[1] P. Azema, R. Valette, and M. Diaz. Petri nets as a common tool for design verification and hardware simulations. In *13th Design Automation Conference*, New-York, 1976.

[2] Albert Benveniste, Bernard Le Goff, and Paul Le Guernic. *Hybrid Dynamical Systems theory and the language* SIGNAL. Research Report 838, INRIA, Rocquencourt, April 1988.

[3] G. Berry and L. Cosserat. *The Esterel programming language and its mathematical semantics.* Research report 327, INRIA FRANCE, 1984.

[4] G. Berry, P Couronné, and G. Gonthier. Synchronous programming of reactive systems: an introduction to ESTEREL. In K. Fuchi and M. Nivat, editors, *Programming of future generation computers*, pages 35–56, ICOT INRIA, North-Holland, 1988.

[5] Patricia Bournai, Vivianne Kerscaven, and Paul Le Guernic. Un environnement graphique pour la conception d'applications temps réel. In *Colloque sur l'ingénierie des interfaces homme-machine*, pages 181–190, Sophia-Antipolis, 1989.

[6] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. LUSTRE: a declarative language for programming synchronous systems. In *14th ACM Symposium on Principles of Programming Languages*, pages 178–188, Munich, 1987.

[7] Thierry Gautier. *Conception d'un langage flot de données pour le temps réel.* PhD thesis, Université de Rennes 1, FRANCE, 1984.

[8] Thierry Gautier, Paul Le Guernic, and Loïc Besnard. SIGNAL: *a declarative language for synchronous programming of real-time systems.* Research report 761, INRIA France, Rocquencourt, November 1987.

[9] Jean-Christophe Gilbon, René Velly, Paul Le Guernic, and Bruno Dutertre. *SOSIE Spécification Objet Synchrone Intégrée.* Rapport de fin de projet, décision d'aide 88 E 0532, Laboratoires de Marcoussis, Marcoussis, Inria/Irisa, Rennes, December 1990.

[10] D. Harel and A. Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*, pages 477–498, Springer-Verlag, New York, 1985.

[11] C. Huizing, R. Gerth, and W. P. de Roever. Modelling STATECHARTS behaviour in a fully abstract way. In M. Dauchet and M. Nivat, editors, *13th Colloquium on Trees in Algebra and Programming CAAP'88 Lecture Notes in Computer Science*, pages 271–294, Springer Verlag, Nancy, France, 1988. Volume 299.

[12] Paul Le Guernic and Albert Benveniste. *Real-time synchronous, data-flow programming: The language* SIGNAL *and its mathematical semantics.* Research report 620, INRIA, Rocquencourt, France, June 1986.

[13] Paul Le Guernic, Albert Benveniste, Patricia Bournai, and Thierry Gautier. SIGNAL*: a data flow oriented language for signal processing.* Research report 378, INRIA France, Rocquencourt, March 1985.

[14] Raymond Marie. *Modélisation par réseaux de files d'attente.* PhD thesis, Université de Rennes 1, 1978.

[15] James L. Peterson. *Petri net theory and the modeling of systems.* Prentice-Hall, 1981.

[16] J. Stankovic and K. Ramamritham. *Hard Real Time Systems. Tutorial,* Computer Society Press of the IEEE, Washington, DC, 1988.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

1991

PI 570 **DESIGN DECISION FOR THE FTM : A GENERAL PURPOSE FAUTL TOLERANT MACHINE**
Michel BANATRE, Gilles MULLER, Bruno ROCHAT, Patrick SANCHEZ
Janvier 1991, 30 pages

PI 571 **ANIMATION CONTROLEE PAR LA DYNAMIQUE**
Georges DUMONT, Parie-Paule GASCUEL, Anne VERROUST
Février 1991, 84 pages

PI 572 **MULTIGRID MOTION ESTIMATION ON PYRAMIDAL REPRESENTATIONS FOR IMAGE SEQUENCE CODING**
Nadia BAAZIZ, Claude LABIT
Février 1991, 48 pages

PI 573 **A SURVEY OF TREE-TRANDUCTIONS**
Jean-Claude RAOULT
Février 1991, 18 pages

PI 574 **THE OPTIMAL ADAPTIVE CONTROL USING RECURSIVE IDENTIFICATION**
Anatolij B. JUDITSKY
Février 1991 - 26 pages

PI 575 **MANUEL SIGNAL**
Patricia BOURNAI, Bruno CHERON, Bernard HOUSSAIS, Paul LE
Paul LE GUERNIC
Février 1991, 84 pages

PI 576 **AN INFORMATION BASED RELIABILITY PREDICTOR FOR SYSTEMS IN OPERATIONAL PHASE**
Kamel SISMAIL
Février 1991 - 22 pages

PI 577 **MULTISCALE STATISTICAL SIGNAL PROCESSING AND RANDOM FIELDS ON HOMOGENEOUS TREES**
Albert BENVENISTE, Michèle BASSEVILLE, Ramine NIKOUKHAH, Alan S. WILLSKY, Ken C. CHOU
Mars 1991 - 18 pages

PI 578 **TOWARDS A DECLARATIVE METHOD FOR 3D SCENE SKETCH MODELING**
Stéphane DONIKIAN, Gérard HEGRON
Mars 1991 - 22 pages

PI 579 **SYSTEMES MARKOVIENS DISCRETS STATIONNAIRES ET APPLICATIONS**
Jean PELLAUMAIL
Mars 1991 - 284 pages

PI 580 **DESCRIPTION ET SIMULATION D'UN SYSTEME DE CONTROLE DE PASSAGE A NIVEAU EN SIGNAL**
Bruno DUTERTRE, Paul LE GUERNIC
Mars 1991 - 66 pages

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique.

ISSN 0249-6399